UiO : Department of Informatics
University of Oslo

The tree-witness ontology rewriting with perfect mappings

Andreas Nakkerud

Master's Thesis, Autumn 2014

Contents

Acknowledgements

First, I would like to thank my advisor, Evgenij Thorstensen. Evgenij has given me great freedom to pursue my own ideas, while always keeping himself up to date on what I have been doing. He could always help me root out flaws in my work, and point me in the right direction when I needed it. His sharp eye and relentless demand for precise reasoning have revealed many failing proofs, and have taught me the valuable lesson that intuitive understanding can only be trusted so far. Evgenij has also been an avid proofreader, helping me improve spelling, chapter ordering, and everything in between.

I would also like to thank my co-advisor Arild Waaler. Arild got me into the topic of this thesis, and his feedback and suggestions have been very important in giving this thesis its direction.

Thanks to Anne Kristine Haugestad and Veronica Øverbye for proof reading this thesis. Despite all my work on this text, they found many small, and some not so small, errors. They also suggested typographical improvements that have made the final product more pleasing to the eye.

I owe much to Roger Antonsen, who introduced to me the field of computer science, and who put me in contact with Arild. Extracurricular work with Roger has been my key motivation for doing a masters thesis in computer science, and his friendship and steady belief in my abilities have helped me through some rough patches.

For their unwavering friendship and support, I extend warm thanks to Leo Karabeg, Mai Amundsen, Anders Hafreager, and Veronica Øverbye. I feel very fortunate to have such good friends outside of my studies.

Finally, I would like to thank Elisabeth Sperrevik Stene. Her loving companionship and support have been the most important things keeping me going at the times when I needed it the most.

Chapter 1

Introduction

This thesis is about a query rewriting called the ontology rewriting. An ontology rewriting of a query is a new query with some set of axioms integrated into it. After an ontology rewriting, we can discard the set of axioms generating that rewriting; their effects should be accounted for in the new query.

Under the Open World Assumption (the assumption that our information may not be complete), the axiom $\texttt{Car} \subseteq \texttt{Vehicle}$ does not state that every answer to the query $\texttt{Car}(x)$ is an answer to the query $\texttt{Vehicle}(x)$. Instead, it states that every answer to $\texttt{Car}(x)$ *should be* an answer to $\texttt{Vehicle}(x)$. In order to account for the axiom $\texttt{Car} \subseteq \texttt{Vehicle}$, we can rewrite the query $\mathbf{q}(x) = \texttt{Car}(x)$ to the query $\mathbf{q}'(x) = \texttt{Car}(x) \cup \texttt{Vehicle}(x)$. Answering $\mathbf{q}(x)$ while respecting that every $\texttt{Car}$ is a $\texttt{Vehicle}$ now yields the same answers as answering $\mathbf{q}'(x)$ directly.

We are interested in cases where the answers to ontology queries are defined in terms of database queries. The link between the ontology and the data base is called the mapping. In order to produce shorter ontology rewritings, we start by absorbing some axioms into the mapping. This process may cause later stages of query answering to take longer, but we can alleviate this effect by optimising the mapping.

We use the tree-witness rewriting algorithm to produce an ontology rewriting. The tree-witness rewriting is very effective when combined with the mapping modification described above. It is also good framework for understanding how to make use of what we call perfect rewritings.

A database query is a perfect rewriting of an ontology query, if the

answers to the database are exactly the answers to the ontology query. A perfect mapping assertion links an ontology query to some perfect rewriting of that query.

Our goal is to use perfect mapping assertions to further improve the tree-witness rewriting.

1.1 Thesis

> "The tree-witness rewriting over H-complete virtual ABoxes [RKZ13] can be improved using perfect mappings [PLL+13]."

In this thesis we argue that we can improve the ontology rewriting presented in [RKZ13], using the notion of perfect mappings introduced in [PLL+13].

In [RKZ13], Rodríguez-Muro et al. present an ontology rewriting based on the tree-witness rewriting over H-complete ABoxes. They introduce a composition of mappings and TBoxes, called $\mathcal{T}$-mappings, that guarantee H-complete virtual ABoxes.

We show how perfect mappings [PLL+13] can be used to further improve the $\mathcal{T}$-mapping by reducing the number of mapping assertions. We also show how perfect mappings can be used to make perfect tree-witnesses, which reduces the size of the tree-witness rewriting.

1.2 Scientific contribution

This work is based on two recent developments [PLL+13, RKZ13] in the area of query answering in Ontology Based Data Access systems. This work is relevant to the Optique project [KJZ+13].

The main scientific contribution of this work is a method for working perfect mappings into the tree-witness rewriting over H-complete ABoxes (see Chapter 6). All the results in Chapter 6 are independent works. Furthermore, the proofs of the theorems of Chapter 5 and the analysis in Section 5.5 are also independent works. An outline of an algorithm implementing our results are presented in Section 6.2.

1.3 Chapter overview

We start with a general introduction to Ontology Based Data Access (OBDA) in Chapter 2. In the last sections of Chapter 2, we explain the context of our work.

In Chapter 3, we give a brief introduction to description logics and ontologies. We define the concept of H-complete ABoxes in Section 3.5.

Chapter 4 deals with mappings: the link between ontologies and their data sources. We also describe how to modify the mapping in order to create an H-complete virtual ABox in Section 4.3. In Section 4.6, we introduce perfect mappings [PLL+13].

In Chapter 5, we describe the tree-witness rewriting, an ontology rewriting over H-complete virtual ABoxes [RKZ13]. We discuss the complexity of the tree-witness rewriting in Section 5.5.

Chapter 6 is where we present most of our independent work. Here we introduce the perfect mapping tree-witness rewriting. We discuss the complexity of the perfect mapping tree-witness rewriting in Section 6.4.

In Section 7.1, we discuss briefly under what circumstances the perfect mapping tree-witness rewriting is most useful. We present our suggestions for future work in Section 7.2.

Chapter 2

Ontology Based Data Access

2.1 Advantages of OBDA

Ontology Based Data Access (OBDA) [Len11], a part of Ontology Based Data Management (OBDM), provides a convenient way to deal with large amounts of data spread over heterogeneous data sources. OBDA allows users to formulate queries in a single user-friendly ontology language. These queries are then unfolded and executed on the data sources.

Many industries and fields of research rely on the ability to handle large amounts of data. As volume, variety and complexity of data increases, access to relevant data becomes more challenging [KJZ+13]. In an ideal system, all data would be stored in user-friendly, homogeneous databases, tailored to the conceptual model of the data. In real life, data will usually be stored in heterogeneous (and possibly outdated) systems, so that experts are needed to help write even simple queries over the data [KJZ+13, KGJ+13].

There is no simple way to avoid heterogeneous data sources. Moving all data to one system can be very expensive and time consuming, and ultimately futile as every new technological advance would require a new move. In addition, there is no guarantee that data gathered for one purpose will be organised in a way that is convenient for all future use of the data. In the real world, we need good ways of dealing with heterogeneous data, and this is where we will apply Ontology Based Data Access.

The key ideas behind Ontology Based Data Management (OBDM) are summarised briefly in [Len11] and in more detail in [PLC+08]. The

goal of OBDM is to separate the conceptual model of data from how that data is stored. This is usually achieved by defining an ontology in some formal language. The terms of these ontologies are then related to database queries. The ontology language can also be used to express axioms about the conceptual model of the data. These axioms allow the ontology language to describe data properties that are not directly reflected in the data sources.

OBDM uses a three layer architecture, with the data source on the bottom and the ontology on top. In the middle we find the mapping between queries over the sources and queries in the ontology language. From a user's perspective, the OBDM architecture provides a single, user-friendly query language, even when the data are stored in very different sources.

When the OBDM scheme only supports reading the source data, we use the term Ontology Based Data Access (OBDA). OBDA is sufficient in applications where the bottom layer (the data sources) are maintained independently of the top level (the query framework). The work in this thesis is limited to OBDA.

2.2 OBDA specifications

The specifications of an OBDA system consists of 4 major parts.

The ontology: The ontology is the user language. This is the language the user will use to pose queries and read answers to queries.

The knowledge base: The knowledge base expresses general facts about the terms of the ontology.

The source schema: The source schema are the schema for the source databases.

The mapping: The mapping links the terminology of the ontology to queries over the source schema.

Example 2.2.1. We want to create an OBDA specification for querying databases containing information about vehicles. In particular, cars and buses. A vehicle is identified by its registration number.

We define the ontology language containing the *concepts* `Vehicle`, `Car` and `Bus`. The knowledge base contains two statements: that every `Car` is a `Vehicle`, and that every `Bus` is a `Vehicle`.

The data source is made up of three databases: an insurance company policy registry, a car and bus dealership inventory, and a bus company inventory.

The insurance company insures only buses, not cars. Their records are stored in the table

$$Policy(regNum, policyNum, amount),$$

where *regNum* is the vehicle registration number, *policyNum* is the policy number, and *amount* is the amount the vehicle is insured for.

The dealership stores their inventory in the table

$$ForSale(regNum, type, price),$$

where *regNum* is the vehicle registration number, *type* is the type of vehicle (car or bus), and *price* is the price of the vehicle.

The bus company stores their vehicles in the table

$$Vehicle(regNum),$$

where *regNum* is the vehicle registration number. The bus company only owns cars and buses.

The mapping links queries over the above tables to the concepts of the ontology. The query `Vehicle`$(regNum)$ is linked to the queries

```
SELECT regNum FROM Policy
SELECT regNum FROM ForSale
SELECT regNum FROM Vehicle
```

That is, every registration number from every database is the registration number of a `Vehicle`.

The query `Car`$(regNum)$ is linked to the query

```
SELECT regNum FROM ForSale WHERE type = 'CAR'
```

Some of the bus company's vehicles are also cars, but we are not able to tell which.

The query `Bus`$(regNum)$ is linked to the queries

```
SELECT regNum FROM ForSale WHERE type = 'BUS'
SELECT Vehicle.regNum, Policy.regNum
   FROM Vehicle, Policy
   WHERE Vehicle.regNum = Policy.regNum
```

In the last query, we have used the knowledge that the bus company only owns cars and buses, and that the insurance company does not insure cars. □

In Chapter 3 we provide a theoretical basis for ontologies and knowledge bases. We describe mappings in Chapter 4. We do not look closer at database schema in this thesis.

2.3 Three-stage query answering

In OBDA, we typically divide query answering into three stages (see [PLC+08]):

(Ontology) rewriting: In this stage we rewrite the initial ontology query. The result is a collection of queries that represent different ways the data we are looking for can be represented. The basis for such a rewriting is the *knowledge base*.

Unfolding: Unfolding is the process of replacing the ontology vocabulary with database queries. This process is not trivial, since many database queries may link to each vocabulary item. We must also deal with differences in variable naming. The result of the unfolding is a collection of queries over the database schema. The *mapping* is the basis for the unfolding.

Execution: Execution of the final queries is a step we leave to the database management systems.

Separating off the execution stage is required whenever we do not have direct access to the data sources. It is also very useful to be able to rely on well established, efficient database management systems.

We separate the ontology rewriting and the unfolding, so that we can deal with the knowledge base and the mapping one at a time. We only consider OBDA specifications where this distinction can be made.

Depending on the size of the ontology and the mapping, the final database query can be very large. It is, however, very likely that the final query contains many redundancies, and can be simplified.

Figure 2.1 provides a simplified view of rewriting and unfolding. The original query is rewritten into three other queries (one of which is

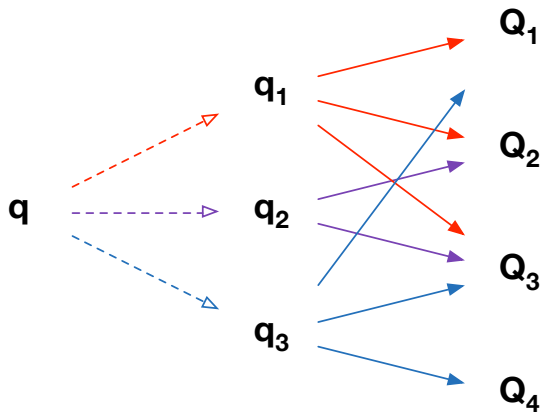

Figure 2.1: Rewriting and unfolding of a query **q**. In the unfolding step (solid arrows), several of the database queries are included more than once. The dashed lines represent the ontology rewriting step.

the original query itself). These queries are then replaced by database queries. In this example, the database reflects some of the relationships that the rewriting was meant to cover, and as such many of the database queries are included more than once. There are a total of eight ways to answer **q**: three ways to answer $\mathbf{q}_1$, two the ways to answer $\mathbf{q}_2$, and three the ways to answer $\mathbf{q}_3$. There is, however, no need to run either database query more than once. In the end, we only execute the four database queries once.

In reality, redundancies are often more complex than simple reuse of queries.

Example 2.3.1. We revisit the OBDA specification of Example 2.2.1, and try to answer the query $\texttt{Vehicle}(regNum)$. Our knowledge base tells us that we must also find the answers to $\texttt{Car}(regNum)$ and $\texttt{Bus}(regNum)$. The mapping now tells us what queries we must execute over the sources. We must execute every query in Example 2.2.1, even though each answer to one of the queries associated with `Car` and `Bus`, is contained in at least one of the queries associated with `Vehicle` (this can be verified using syntactic query containment checks). □

The topic of this thesis is a method for simplifying the rewriting step (dashed arrows of Figure 2.1) at the cost of expanding the mapping, potentially increasing the cost of the unfolding step (solid arrows of

Figure 2.1). We address the last issue by presenting methods for simplifying the mapping, which reduce the cost of the unfolding.

2.4 Our contribution: The tree-witness rewriting with perfect mappings

Our work focuses on the ontology rewriting stage of OBDA. We work with the tree-witness rewriting, an ontology rewriting presented in [KKZ12]. We use a version of the tree-witness rewriting that works on H-complete mappings (Chapter 5), introduced in [RKZ13]. An H-complete mapping is a mapping that accounts for many of the knowledge base axioms. We can make a mapping H-complete by extending it according to the knowledge base axioms (Section 4.3).

Making a mapping H-complete is not entirely a one-time process, but it only needs to be redone when either the knowledge base (ontology) or the mapping changes. When the mapping has been modified, the ontology rewriting step can be achieved faster. Since many of the properties in the knowledge base are now reflected by the mapping, they do not need to enter into the ontology rewriting.

As mentioned in the discussion following Figure 2.1, increasing the mapping is done at the expense of the unfolding step. In order to counteract this, we also analyse the mapping and make simplifications where possible. We will not look into the unfolding step, aside from discussing simplification of the mapping (see Section 7.2.2 for a brief discussion of further studies into unfolding).

We attempt to improve on the tree-witness rewriting by introducing perfect mapping assertions [PLL+13] into it (Section 4.6). A perfect mapping assertion links an ontology query to a complete (perfect) ontology rewriting and unfolding of that query.

Making the mapping H-complete is a very useful first step. It makes one-time improvements to the mapping, which then lower the cost of rewriting subsequent queries with the tree-witness rewriting. We discuss this trade-off in Sections 4.5 and 6.4. The perfect mapping approach does not share the advantage of one-time improvements. We discuss the performance of the tree-witness rewriting with perfect mappings in Section 6.4.

Chapter 3

Ontologies

An ontology is a description of some domain of discourse. The ontologies we will be working with are built up of concepts, roles, and individuals.

The individuals are the objects being described by the ontology, the concepts are groups of individuals that have something in common, and the roles describes relationships between individuals. Using an analogy to first order logic, we could think of individuals as domain elements or constants, concepts as atomic relations, and roles as binary relations.

In order to formalise knowledge about our ontology, we use an ontology language. We focus our attention on the Description logic dialect OWL 2 QL (originally introduced as DL-$Lite_R$) [CDL+06, CDL+07, W3C14].

There are many candidate languages for the formulation of ontologies. First order logic (FOL) is one of the most well known, but FOL is not decidable. We do not need the full expressiveness of FOL. Instead, we will define our ontologies using Description logics. We will only work with the Description logics that are decidable FOL fragments.

3.1 Notes on notation

In much of the theoretical development we do not distinguish between symbols and actual objects. For example, we may use a both as the formal symbol for an individual, and to represent that same individual as a member of an abstract knowledge base. We will make the distinction clear when the two meanings may be confused.

3.2 Description logic

Ontologies are commonly described using Description logic. Description logics express axioms about *concepts*, *roles* and *individuals*. Concepts are Description logic versions of unary properties, while roles are binary relations.

Definition 3.2.1 (Syntax of Description logic)**.** Let C_1 and C_2 be concepts, R_1 and R_2 roles, and a_1 and a_2 individuals. The following are the syntactic elements of Description logic that we will be using:

$\top$ universal concept,

$\bot$ empty concept,

$C_1 \sqcap C_2$ concept intersection (or conjunction),

$C_1 \sqcup C_2$ concept union (or disjunction),

$\neg C_1$ concept negation (or complement),

$\exists R_1.C_1$ existential restriction,

$\exists R_1.\top$ universal restriction,

R_1^- inverse role,

$C_1 \sqsubseteq C_2$ concept inclusion,

$R_1 \sqsubseteq R_2$ role inclusion,

$C_1(a_1)$ concept assertion, and

$R_1(a_1, a_2)$ role assertion.

□

Some Description logics are more expressive, and have a richer syntax than that of Definition 3.2.1. The Description logics we use will be allowed to use some subsets of the above syntax.

Definition 3.2.2 (Signatures of Description logic languages)**.** A *signature* of a Description logic language is an ordered triple $\langle \Gamma_C, \Gamma_R, \Gamma_I \rangle$ where Γ_C, Γ_R, and Γ_I are pairwise disjoint alphabets of concept names, role names, and individual names, respectively. □

The alphabet of individual names Γ_I will often be divided into constant names and variable names. The constant names are names for actual individuals, whereas the variable names are names that can be used to represent any individual. Constant and variable names work in much the same way as constants and variables of first order logic.

3.3 Interpretations of Description logic

Interpretations, or terminological interpretations, of Description logic are similar to models of first-order logic. Interpretations of Description logic consist of a *domain* (or *universe*), and interpretations of the concepts and roles. The concepts are interpreted as subsets of the domain, while the roles are interpreted as sets of pairs over the domain.

Definition 3.3.1 (Semantics of Description logic)**.** Let $\mathcal{O} = \langle \Gamma_C, \Gamma_R, \Gamma_I \rangle$ be the signature of a Description logic language, or simply *an ontology*. An *interpretation* $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ over $\langle \Gamma_C, \Gamma_R, \Gamma_I \rangle$ consists of a non-empty set $\Delta^{\mathcal{I}}$, called the *domain*, and a function $\cdot^{\mathcal{I}}$ such that

- for every $a \in \Gamma_I$, $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$,
- for every $C \in \Gamma_C$, $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, and
- for every $R \in \Gamma_R$, $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

Furthermore, if C_1 and C_2 are concepts, R_1 and R_2 roles, and a_1 and a_2 individuals, then

- $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$,
- $\bot^{\mathcal{I}} = \emptyset$,
- $(C_1 \sqcap C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$,
- $(C_1 \sqcup C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$,
- $(\neg C_1)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C_1^{\mathcal{I}}$,
- $(\exists R_1.C_1)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists y \in C_1^{\mathcal{I}} \text{ such that } \langle x, y \rangle \in R_1^{\mathcal{I}}\}$, and
- $(R_1^-)^{\mathcal{I}} = \{\langle x, y \rangle \mid \langle y, x \rangle \in R_1^{\mathcal{I}}\}$.

Concept and role inclusions and concept and role assertions form the atomic statements of Description logic. Let ϕ be a formula of Description logic. We define $\mathcal{I} \models \phi$, read as $\mathcal{I}$ models ϕ, as follows:

- $\mathcal{I} \models C_1 \sqsubseteq C_2$ if and only if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$,
- $\mathcal{I} \models R_1 \sqsubseteq R_2$ if and only if $R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$,
- $\mathcal{I} \models C_1(a_1)$ if and only if $a_1^{\mathcal{I}} \in C_1^{\mathcal{I}}$, and
- $\mathcal{I} \models R_1(a_1, a_2)$ if and only if $\langle a_1^{\mathcal{I}}, a_2^{\mathcal{I}} \rangle \in R_1^{\mathcal{I}}$.

We will only look at conjunctions of atomic Description logic formulas, in which case $\mathcal{I} \models \phi \wedge \psi$ if and only if $\mathcal{I} \models \phi$ and $\mathcal{I} \models \psi$.

More generally, if Γ is a set of axioms, then $\mathcal{I} \models \Gamma$ if and only if $\mathcal{I} \models \phi$ for every $\phi \in \Gamma$. □

3.4 Ontology axioms

In general, any Description logic formula can be an ontology axiom. In order to make reasoning over our ontologies efficient, we limit what formulas we allow as axioms. First off, we only allow atomic statements. Variations of Description logic differ in what axioms they allow (see [Rud11] for a list of common Description logics and the convention for naming them). We will focus on variations of DL-Lite, a set of Description logics where answering of conjunctive queries is manageable for very large databases [PLC+08].

The approach to ontology rewriting known as the *tree-witness rewriting* depends on restricting the ontology axioms to follow OWL 2 QL (see [KKPZ13]).

Definition 3.4.1 (OWL 2 QL knowledge base)**.** Given individual names a_i, concept names A_i, and role names P_i, we define basic concepts B and basic roles R as

$$R ::= P_i \mid P_i^-$$
$$B ::= \bot \mid A_i \mid \exists R,$$

where $\exists R$ is shorthand for $\exists R.\top$. Let $\mathcal{T}$ (the TBox) be a finite set of *inclusion axioms* of the form

$$B_1 \sqsubseteq B_2, \qquad B_1 \sqcap B_2 \sqsubseteq \bot, \qquad R_1 \sqsubseteq R_2, \qquad R_1 \sqcap R_2 \sqsubseteq \bot,$$

and let $\mathcal{A}$ (the ABox) be a finite set of *assertion axioms* of the form

$$A_m(a_i), \qquad P_m(a_i, a_j).$$

The pair $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is a *knowledge base*. The restrictions placed on the TBox axioms determine the complexity of reasoning over the knowledge base. We write $\text{ind}(\mathcal{A})$ for the set of individual names appearing in $\mathcal{A}$.

We also allow TBox axioms $B_i \equiv B_j$, understanding that they are shorthand for $B_i \sqsubseteq B_j$ and $B_j \sqsubseteq B_i$. Similarly, we allow $R_i \equiv R_j$. □

Sometimes we refer to knowledge bases in place of ontologies. We then use the knowledge base to represent the ontology containing the concept, role and individual names present in the knowledge base.

Theorem 3.4.2. *In addition to the TBox axioms in Definition 3.4.1, we can allow axioms on the form $B_1 \sqsubseteq \exists R.B_2$ in OWL 2 QL (see [KKZ12]). Axioms on this form are shorthand for the following three axioms*

$$B_1 \sqsubseteq \exists R_B, \qquad \exists R_B^- \sqsubseteq B_2, \qquad R_B \sqsubseteq R,$$

where R_B is a new role name. The new role R_B has a domain containing B_1, a range contained in B_2, and is contained in R.

Proof. We omit global universal quantifiers for brevity. The shorthand version of the axiom can be written in first-order logic notation as

$$B_1^{\mathcal{I}}(x) \rightarrow \exists y[R^{\mathcal{I}}(x, y) \wedge B_2^{\mathcal{I}}(y)], \tag{3.1}$$

for every interpretation $\mathcal{I}$.

The expanded version of this axiom can be written as the conjunction of the three formulas

$$B_1^{\mathcal{I}}(x) \rightarrow \exists y[R_B^{\mathcal{I}}(x, y)], \quad \exists x R_B^{\mathcal{I}}(x, y) \rightarrow B_2^{\mathcal{I}}(y), \quad R_B^{\mathcal{I}}(x, y) \rightarrow R^{\mathcal{I}}(x, y), \tag{3.2}$$

for every interpretation $\mathcal{I}$.

It is straightforward to show that (3.2) implies (3.1): Assume the left-hand side of (3.1), the right-hand side follows by the implications of (3.2). To show the other direction, we observe that R_B is a new role name (one that does not appear in other axioms, either in the ABox or the TBox). As a consequence, we can let $R_B^{\mathcal{I}} = R^{\mathcal{I}} \cap (B_1^{\mathcal{I}} \times \Delta^{\mathcal{I}})$, and then (3.2) holds. □

It is possible to create axioms that are not consistent with each other. That is, axioms that cannot be true in the same interpretations.

Definition 3.4.3 (Interpretation of a knowledge base)**.** Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a knowledge base, and let $\mathcal{I}$ be an interpretation. If $\mathcal{I} \models \mathcal{T}$ and $\mathcal{I} \models \mathcal{A}$, we say that $\mathcal{I}$ *is a model for* $\mathcal{K}$, and write $\mathcal{I} \models \mathcal{K}$. $\mathcal{K}$ is *consistent* if it has a model. □

We will need the models of knowledge bases when we define query answers. We will be particularly interested in the assertions that must be true in any model for some knowledge base. These valid assertions form the basis for certain query answers (Definition 3.6.6).

3.5 Ontology reasoning

The purpose of the TBox is to add implicit information to our knowledge base. TBox axioms allow us to infer facts about individuals that are not stored explicitly in the ABox.

In addition to containing the key to implicit facts about the ABox, the TBox can also contain implicit facts about itself. These implicit facts follow from explicit inclusion axioms and the transitivity of inclusion. The transitivity of inclusion follows from the semantics of Description logic, Definition 3.3.1.

Definition 3.5.1 (TBox induced subsumption relation)**.** Let $\mathcal{T}$ be a TBox. The relation $\sqsubseteq_{\mathcal{T}}$ is the *subsumption relation induced by* $\mathcal{T}$, defined so that

$$\begin{aligned} B_1 \sqsubseteq_{\mathcal{T}} B_2 \quad &\text{iff} \quad \mathcal{T} \models B_1 \sqsubseteq B_2 \\ R_1 \sqsubseteq_{\mathcal{T}} R_2 \quad &\text{iff} \quad \mathcal{T} \models R_1 \sqsubseteq R_2. \end{aligned}$$

where $\mathcal{T} \models \phi$ if $\mathcal{I} \models \phi$ for every interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{T}$. □

With OWL 2 QL axioms, the induced inclusions all follow from the transitivity of inclusion.

Definition 3.5.2 (Valid formulas)**.** Let $\mathcal{K}$ be a knowledge base and ϕ a Description logic formula. If $\mathcal{I} \models \phi$ for every interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{K}$, then we say that ϕ *is valid in for* $\mathcal{K}$, and write $\mathcal{K} \models \phi$. Let Γ be a set of concept and role assertions. If for every $\mathcal{I} \models \mathcal{K}$, we have $\mathcal{I} \models \Gamma$, then $\mathcal{K} \models \Gamma$. □

In the above definition of $\mathcal{K} \models \Gamma$, we required that every interpretation is a model for the entire set of atoms, rather than each atom being modelled individually by each interpretation. This distinction will be important when we introduce variables, as we will require a uniform assignment to the variables of a query. (Without variables the two definitions are equivalent.)

We are now ready to define H-completeness. A knowledge base with an H-complete ABox is a knowledge base where all valid concept and role assertion are stated explicitly in the ABox.

Definition 3.5.3 (H-complete ABoxes)**.** Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a knowledge base. The Abox $\mathcal{A}$ is *H-complete with respect to* $\mathcal{T}$ if

1.	$R_1(a_1, a_2) \in \mathcal{A}$	if	$R_2(a_1, a_2) \in \mathcal{A}$ and $R_2 \sqsubseteq_{\mathcal{T}} R_1$
2.	$A_1(a_1) \in \mathcal{A}$	if	$A_2(a_1) \in \mathcal{A}$ and $A_2 \sqsubseteq_{\mathcal{T}} A_1$
3.	$A_1(a_1) \in \mathcal{A}$	if	$R_1(a_1, a_2) \in \mathcal{A}$ and $\exists R_1 \sqsubseteq_{\mathcal{T}} A_1$

for all roles R_i and concept names A_i. We write $R_i(a_1, a_2) \in \mathcal{A}$ if $R_i = P_j$ and $P_j(a_1, a_2) \in \mathcal{A}$, or if $R_i = P_j^-$ and $P_j(a_2, a_1) \in \mathcal{A}$. A knowledge base is H-complete if its ABox is H-complete. □

3.6 Queries and variables

Informally, a query is a set of restrictions on a set of variables. In Description logic, any formula built from atomic ontology language statements (Definition 3.2.1) can be an ontology query. We will limit ourselves to conjunctive queries with existentially quantified variables.

Before we formally define queries, we need to take a close look at the individual names of Definition 3.2.2.

Definition 3.6.1 (Variable names and domain individual names)**.** We divide the set Γ_I into two disjoint sets Γ_O and Γ_V. Γ_O contains an infinite number of individual names for use in ABoxes, and names for every individual $x \in \Delta^{\mathcal{I}}$ in the domain of every interpretation. Γ_V contains an infinite set of variable names. □

Definition 3.6.2 (Conjunctive query)**.** Let $\mathcal{O} = \langle \Gamma_C, \Gamma_R, \Gamma_I \rangle$ be an ontology, and let $Y \subseteq \Gamma_V$ and $X \subseteq \Gamma_O \cup \Gamma_V$ be two disjoint sets of individual names. Let $\phi(x_1 \ldots x_n, y_1 \ldots y_m)$ be a conjunction of atomic

statements with names from $\Gamma_C \cup \Gamma_R$, in which the individual names are all in $X \cup Y$. The formula

$$\Phi = \exists y_1 \dots y_m \phi(x_1, \dots, x_n, y_1, \dots, y_m)$$

is a *conjunctive query (over $\mathcal{O}$)*. We will often use $\exists \vec{y} \phi(\vec{x}, \vec{y})$ or $\exists \vec{y} \phi(x_1 \dots x_n, \vec{y})$ as shorthand notation for the above query.

The variables in X are called the distinguished variables (or the answer variables), the variables in Y are called the non-distinguished variables. The set of the distinguished variables of Φ is denoted $\mathrm{dvar}(\Phi)$, the set of the non-distinguished variables of Φ is denoted $\mathrm{nvar}(\Phi)$. The set all the variables of Φ is denoted $\mathrm{var}(\Phi)$. □

We will sometimes refer to queries over knowledge bases. What we mean by this is queries over the ontology the knowledge base is formulated in.

Before we go on, we need to define the following notation.

Definition 3.6.3 (Function restriction)**.** Let $f : X \to Y$ be a function with domain X and range Y, and A a subset of X. Then

$$f_A = f \restriction A = f \cap (A \times Y)$$

is the *restriction of f to A*. □

The answers to a query are defined in terms of assignments to the distinguished variables of the query. Informally, an assignment is the answer to a given query, if the assignment makes the query formula true.

Definition 3.6.4 (Individual name substitution)**.** Let $X \subseteq \Gamma_V$ be some set of variables, and $\sigma : X \to \Gamma_O$ a function from X to the individual names Γ_O. Then σ is an *individual name substitution*, or just *substitution*, when no confusion may arise, and the formula $\phi\sigma$ is the formula ϕ with every occurrence of $x \in X$ replaced by $\sigma(x)$. □

We will often be interested in the restriction of some individual name substitution σ to the answer variables of some query Φ. The individual name substitution $\sigma_{\mathrm{dvar}(\Phi)}$ is exactly this restriction.

Given the above definition of individual name substitutions, we are now in a position to give a formal definition of query answers.

Definition 3.6.5 (Semantics of Description logic with variables)**.** Let $\mathcal{I}$ be an interpretation and $\Phi = \exists y_1 \dots y_m \phi(a_1, \dots, a_n, y_1, \dots, y_m)$ be a conjunctive query where a_i are individual names. Then $\mathcal{I} \models \Phi$ if there is some substitution μ such that $\mathcal{I} \models \phi(a_1, \dots, a_n, y_1, \dots, y_m)\mu_{\{y_1,\dots,y_m\}}$.

We also have that $\mathcal{K} \models \Phi$, if and only if $\mathcal{I} \models \Phi$ for every $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{K}$. □

We will mostly be interested in the answers that are valid (i.e. true in all interpretations) for some knowledge base (see the discussion after Definition 3.4.3).

Definition 3.6.6 (Certain answers)**.** Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a knowledge base, and $\Phi = \exists \vec{y} \phi(x_1 \dots x_n, \vec{y})$ a conjunctive query. The *certain answers to* Φ *over* $\mathcal{K}$ is the set $CertAns(\Phi, \mathcal{K})$ defined by:

$$\sigma_{\{x_1 \dots x_n\}} \in CertAns(\Phi, \mathcal{K}) \qquad \text{if} \qquad \mathcal{K} \models \Phi\sigma_{\{x_1 \dots x_n\}},$$

where $\sigma_{\{x_1 \dots x_n\}}$ is a substitution restricted to the distinguished variables of the conjunctive query Φ.

If $\Phi = \{\Psi_1 \dots \Psi_k\}$ is a union of conjunctive queries, then

$$CertAns(\Phi, \mathcal{K}) = \bigcup_{i=1}^{k} CertAns(\Psi_i, \mathcal{K}),$$

where Ψ_i are conjunctive queries with the same distinguished variables. □

In short, the certain answers to an ontology query are the answers dictated by the knowledge base.

We will frequently use sets of atomic queries to represent conjunctions. However, in Definition 3.6.6, we interpret a set of queries as a disjunction. We are mostly interested in Unions of Conjunctive Queries, or queries that can be expressed on Disjunctive Normal Form. In this case, sets of atoms will be conjunctions, while sets of sets will be disjunctions. We will specify our intended meaning whenever it is not clear from the context.

Theorem 3.6.7. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a knowledge base, and $\Phi = \exists \vec{y} \phi(\vec{x}, \vec{y})$ a conjunctive query. If $\sigma \in CertAns(\Phi, \mathcal{K})$, then σ assigns a member of $ind(\mathcal{A})$ (an ABox individual) to every variable in $\vec{x}$.*

Proof. Let $\Phi = \exists \vec{y} \phi(\vec{x}, \vec{y})$ be a conjunctive query over a knowledge base $\mathcal{K}$, with an answer σ that maps some variable in $\vec{x}$ to an individual c, such that c does not appear in the ABox. We prove that σ is not a certain answer to Φ.

Let $\mathcal{I}$ be an interpretation such that $\mathcal{K} \models \mathcal{I}$ and $\mathcal{I} \models \Phi\sigma$. Let $\mathcal{I}'$ be the interpretation $\mathcal{I}$ with every occurrence of c replaced by a new constant c'. In $\mathcal{I}'$, c does not occur in any concept or role assertion, and $\mathcal{I}' \not\models \Phi\sigma$. Since $\mathcal{I}$ and $\mathcal{I}'$ are equivalent up to constant renaming, $\mathcal{K} \models \mathcal{I}'$, so σ is not a *certain* answer to Φ. □

3.7 Ontology rewriting

Definition 3.6.6 tells us that the (certain) answers to a query are those answers that are dictated by the knowledge base, i.e. the answers that every interpretation agrees upon.

Answering queries over a knowledge base with an empty TBox is trivial. The certain answers to a query ϕ over an ABox are the substitutions σ such that every atom of $\phi\sigma$ is an assertion in the ABox.

Definition 3.7.1 (Ontology rewriting)**.** Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a knowledge base, and Φ a conjunctive query. The query Φ_0 is an *ontology rewriting of* Φ *and* $\mathcal{T}$ if

$$CertAns(\Phi_0, \langle \emptyset, \mathcal{A} \rangle) = CertAns(\Phi, \mathcal{K})$$

for any ABox $\mathcal{A}$. □

Finding the ontology rewriting of a query over a knowledge base is one of the most common ways of answering ontology queries, but the ontology rewriting can be very large. The ontology rewriting is a union of conjunctive queries that together encode all the relevant information from the TBox.

Definition 3.7.2 (Ontology rewriting over H-complete ABoxes)**.** Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a knowledge base, and Φ a conjunctive query. Φ_0 is an *ontology rewriting of* Φ *and* $\mathcal{T}$ *for H-complete ABoxes* if

$$CertAns(\Phi_0, \langle \emptyset, \mathcal{A} \rangle) = CertAns(\Phi, \mathcal{K})$$

for any H-complete ABox $\mathcal{A}$. □

Ontology rewritings over H-complete ABoxes are often smaller and simpler than general ontology rewritings, because they do not need to take into account inclusion axioms where both concepts (or roles) are named, or where the including concept is named. This is because the statements of these axioms are reflected in the ABox (see Definition 3.5.3). The only inclusion axioms that are not guaranteed to be reflected by an H-complete ABox are

$$A \sqsubseteq \exists R, \qquad \exists R_1 \sqsubseteq \exists R_2.$$

The tree-witness rewriting (Chapter 5) is one way to deal with these remaining axioms.

Chapter 4

Mappings and virtual ABoxes

The mapping is an important part of the OBDA architecture. It is the mapping that defines the content of the ABox, and thus the answers to our queries. In our ontology rewriting algorithm, we will modify the mapping in order to deal with some of the axioms in the TBox. It is therefore necessary to have a good understanding of mappings, and how they contribute answers.

4.1 Mappings

Using mappings is a convenient way of specifying the content of an ABox. Mapping assertions define ontology concepts and roles in terms of queries over the data source.

Definition 4.1.1 (Mapping assertions and mappings)**.** Let $\mathcal{S}$ be a database schema, $\mathcal{O}$ an ontology, $Q(\vec{x}, \vec{y})$ a conjunctive query over $\mathcal{S}$, and $\mathbf{q}(\vec{x})$ an atomic query over $\mathcal{O}$. The statement

$$m : Q(\vec{x}, \vec{y}) \rightsquigarrow \mathbf{q}(\vec{x}),$$

is a *(GAV) mapping assertion from $\mathcal{S}$ to $\mathcal{O}$*, where m is the name, $Q(\vec{x}, \vec{y})$ is the body, and $\mathbf{q}(\vec{x})$ is the head of the mapping assertion. A set of (GAV) mapping assertions from a database schema $\mathcal{S}$ to an ontology $\mathcal{O}$ is denoted $\mathcal{M}$, and called a *(GAV) mapping from $\mathcal{S}$ to $\mathcal{O}$*. □

We make sure each mapping assertion in a mapping uses unique variable names. We omit the mapping name m where convenient. GAV stand for Global As View. A GAV mapping assertion is a mapping assertion where there are no non-distinguished variables in the head, and where every

distinguished variable in the head is also a distinguished variable in the body.

Note that we in Definition 4.1.1 refer to queries over ontologies, not over knowledge bases. Here, this distinction is important. Since there is no ABox in our knowledge base, the ontology is the source of usable names.

Because we are using mappings to populate virtual ABoxes, the knowledge base may not contain all the concept, role, and individual names we are allowed to use in queries.

In GAV mappings, a mapping assertion with a conjunctive query as its head is simply a shorthand way of writing one assertion for each conjunct, each with the same body as the original assertion. From here on we assume that the heads of ordinary GAV mappings are atomic (we will make an exception for perfect mappings in Chapter 6).

4.1.1 Database query answers

A typical database system provides answers in the form of tuples. These tuples contain one entry for each (non-existential, or distinguished) variable in the query. Since we use substitutions to represent answers, we define the certain answers to a database query in the following way.

Definition 4.1.2 (Database query answers)**.** Let $\mathcal{S}$ be a database schema, D be an instance of $\mathcal{S}$, and $Q(\vec{z})$ a query over $\mathcal{S}$. The set $Ans(Q(\vec{z}), D)$ is the set of tuples $\vec{a}$ resulting from executing $Q(\vec{z})$ over D.

We extend the notion of certain answers to database queries in the following way:

$$CertAns(Q(\vec{z}), D) = \{\sigma^{\vec{a}} \mid \vec{a} \in Ans(Q(\vec{z}), D)\},$$

where $\sigma^{\vec{a}}$ is the individual name substitution (Definition 3.6.4) mapping the variables in $\vec{z}$ to the corresponding values in the answer $\vec{a}$. □

4.1.2 Substitution and unification

We will often encounter queries whose atoms match the heads of mapping assertions down to variable renaming or substitution of constants for variables. We need a way to decide what mapping assertions are applicable to a given query atom (or subquery in the case of perfect mappings).

Definition 4.1.3 (Substitution)**.** A *substitution* is a set $\sigma = \{x_i \mapsto t_i\}$, where the x_i are distinct variables and the t_i are terms. $\phi\sigma$ is the expression ϕ with every occurrence of x_i replaced by t_i. □

The individual name substitutions (Definition 3.6.4) are substitutions where the terms are limited to constants.

Using a substitution, we can make a general query match a more specific query. Unification [Llo84] is used in a similar way as substitutions, but unification is less strict. When unifying two queries, we do not care which one is more general.

Definition 4.1.4 (Unification)**.** Let ϕ and ψ be expression and σ a substitution. If $\phi\sigma = \psi\sigma$, then σ *unifies*, and is a *unifier* for, ϕ and ψ. □

Unification is not a unique process, but there are some unifiers that are better (more general) than others.

Definition 4.1.5 (Most general unifier)**.** Let ϕ and ψ be expressions. A *most general unifier (MGU)* for ϕ and ψ is a unifier σ such that whenever $\phi\tau = \psi\tau$, τ is a composition of σ and some substitution μ.

MGUs are unique up to variable renaming (variants). □

The MGUs are the unifiers we need for unfolding. If we do not restrict our unifiers to MGUs, we risk losing answers by substituting variables for constants or already used variables when this is not needed. For more on MGUs, see for example [Llo84, MM82].

4.2 Query answering with mappings

There are several ways to answer a query using mappings. [PLC+08] suggests two approaches: the bottom-up approach, and the top-down approach.

In the bottom-up approach, we use the mapping to populate an ABox. We then answer ontology queries using this new ABox. The bottom-up approach is conceptually simple, but very expensive for large data sources.

In the top-down approach, we avoid populating the ABox, and let the mapping form the basis for a *virtual* ABox. Starting with the ontology rewriting (Definition 3.7.1) of a query, we find some database query that we can use to find the virtual ABox answers.

Definition 4.2.1 (Virtual ABox)**.** A virtual ABox is an ABox whose assertions are defined by a mapping. □

The *unfolding* of a query is a union (disjunction) of every possible combination of mapping assertions that can be applied to the atoms of the query.

Definition 4.2.2 (Unfolding and mapping rewriting)**.** Let $\mathcal{O}$ be an ontology, $\mathcal{S}$ a database schema, $\mathcal{M} = \{m_i : Q_i(\vec{x}_i, \vec{y}_i) \rightsquigarrow \mathbf{q}_i(\vec{x}_i)\}$ a mapping from $\mathcal{S}$ to $\mathcal{O}$, and $\mathbf{cq}(\vec{z})$ a conjunctive query over $\mathcal{O}$ with atoms $S_i(\vec{z}_i)$, where $\vec{z}_i$ contains the variables in $\vec{z}$ occurring in S_i.

If there is an MGU σ unifying each $S_i(\vec{z}_i)$ with the head of some mapping assertion m_i, then let $M \subseteq \mathcal{M}$ be the set if these mapping assertions. The query

$$\left(\bigwedge_{m_i \in M} Q_i(\vec{x}_i, \vec{y}_i) \right) \sigma$$

is a *mapping rewriting of* $\mathbf{cq}(\vec{z})$ *and* $\mathcal{M}$ *based on* σ *and* M. The *unfolding of* $\mathbf{cq}(\vec{z})$ *over* $\mathcal{M}$ is the union of all mapping rewritings of $\mathbf{cq}(\vec{z})$ and $\mathcal{M}$.

If we need to use a mapping assertion more than once (because some concept or role name appears more than once), we make one copy of the relevant mapping assertion for each application, with a new name and new variable names. □

Definition 4.2.2 shows how we can overcome variable naming differences and variable name reuse (equality restrictions) in order to answer conjunctive queries using mappings. We start by giving every variable a unique name, then allow the substitution σ to reflect equality constraints.

Example 4.2.3. We unfold the query $R(x, y) \wedge R(y, z)$ using the mapping

$$m \quad : \quad Q(x_1, x_2) \rightsquigarrow R(x_1, x_2).$$

Since we need to apply m twice, we make a copy

$$m' \quad : \quad Q(y_1, y_2) \rightsquigarrow R(y_1, y_2).$$

The only mapping rewriting is

$$\Big(Q(x_1, x_2) \wedge Q(y_1, y_2)\Big)\{x_1 \mapsto x,\ x_2 \mapsto y,\ y_1 \mapsto y,\ y_2 \mapsto z\},$$

which is then also the unfolding of the query. □

An Ontology Based Data Access system is defined by an ontology, a TBox defining properties of the ontology, a data source schema, and the mapping defining the connection between the ontology and the data source.

Definition 4.2.4 (OBDA specification)**.** Let $\mathcal{O}$ be an ontology, $\mathcal{S}$ a database schema, $\mathcal{T}$ a TBox over $\mathcal{O}$, and $\mathcal{M}$ a mapping from $\mathcal{S}$ to $\mathcal{O}$. The tuple

$$\mathcal{B} = \langle \mathcal{O}, \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$$

is an *OBDA specification*. A query over $\mathcal{O}$ is also a query over $\mathcal{B}$. □

Using the unfolding from Definition 4.2.2, we can define the answers to a query over an OBDA specification.

Definition 4.2.5 (Certain answers from unfolding)**.** Let $\mathcal{B} = \langle \mathcal{O}, \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$ be an OBDA specification, D an instance of $\mathcal{S}$, and $\mathbf{cq}(\vec{x})$ a conjunctive query over $\mathcal{B}$. Let $\mathbf{cq}_0(\vec{x})$ be an ontology rewriting of $\mathbf{cq}(\vec{x})$. The *certain answers to* $\mathbf{cq}(\vec{x})$ *over* $\mathcal{B}$ *and* D, denoted $CertAns(\mathbf{cq}(\vec{x}), \mathcal{B}, D)$, is the set of all substitutions $\sigma_{x_1,\ldots,x_n}$, such that $\sigma = \mu\nu$ where

1. there is some mapping rewriting $CQ(\vec{x}, \vec{y})$ of $\mathbf{cq}_0(\vec{x})$ and $\mathcal{M}$ based on μ and some $M \subseteq \mathcal{M}$, and
2. $\nu \in CertAns(CQ(\vec{x}, \vec{y}), D)$.

That is, each certain answer to a conjunctive query over an OBDA specification and a database instance, is the composition of two unifiers: the unifier producing some mapping rewriting, and the answers to that mapping rewriting. The certain answer is the composition, restricted to the variables of the original query. □

Example 4.2.6. We look at the query $\mathbf{q}(x, y) = R(x, y) \wedge S(y, z)$, with mapping

$$\begin{array}{lcll} m_1 & : & Q_R(x_1, x_2) & \rightsquigarrow \quad R(x_1, x_2) \\ m_2 & : & Q_S(y_1, y_2) & \rightsquigarrow \quad S(y_1, y_2). \end{array}$$

We unify the heads of m_1 and m_2 with the atoms of $\mathbf{q}$ using the unifier

$$\mu = \{x \mapsto x_1, \quad x_2 \mapsto y_1, \quad y \mapsto y_1, \quad y_2 \mapsto z\},$$

resulting in the mapping rewriting

$$CQ = Q_R(x_1, y_1) \wedge Q_S(y_1, z).$$

Note that μ may not be the simplest unifier. We have chosen μ to illustrate that no variables need to be preferred over others. Assume

$$\sigma = \{x_1 \mapsto a, \quad y_1 \mapsto b, \quad z \mapsto c\}$$

is a certain answer to CQ, then

$$\begin{aligned}(\mu\sigma)_{\{x,y\}} &= \big(\{x \mapsto a, \quad x_2 \mapsto b, \quad y \mapsto b, \quad y_2 \mapsto c, \quad z \mapsto c\}\big)_{\{x,y\}} \\ &= \{x \mapsto a, \quad y \mapsto b\}\end{aligned}$$

is a certain answer to $\mathbf{q}(x, y)$ (where we have restricted $\mu\sigma$ to the answer variables x and y). □

4.3 H-complete virtual ABoxes

In Chapter 3, we looked at how to make an ABox H-complete. We need a way, analogous to Definition 3.5.3, in which to make a virtual ABox H-complete. Rodríguez-Muro et al. [RKZ13] deal with H-completeness of the virtual ABox by defining the composition of a TBox and a mapping.

Definition 4.3.1 ($\mathcal{T}$-mapping)**.** Let $\mathcal{O}$ be an ontology, $\mathcal{S}$ a database schema, $\mathcal{M}$ a mapping from $\mathcal{S}$ to $\mathcal{O}$, and $\mathcal{T}$ a TBox over $\mathcal{O}$. The *$\mathcal{T}$-mapping composed of $\mathcal{M}$ and $\mathcal{T}$*, denoted $\mathcal{M}^{\mathcal{T}}$ is defined such that $\mathcal{M} \subseteq \mathcal{M}^{\mathcal{T}}$, and

1. if $Q(x_1, x_2, \vec{y}) \rightsquigarrow R_1(x_1, x_2)$ is in $\mathcal{M}$ and $R_1 \sqsubseteq_{\mathcal{T}} P_1$, then $Q(x_1, x_2, \vec{y}) \rightsquigarrow P_1(x_1, x_2)$ is in $\mathcal{M}^{\mathcal{T}}$;
2. if $Q(x_1, x_2, \vec{y}) \rightsquigarrow R_1(x_1, x_2)$ is in $\mathcal{M}$ and $\exists R_1 \sqsubseteq_{\mathcal{T}} A_1$, then $Q(x_1, x_2, \vec{y}) \rightsquigarrow A_1(x_1)$ is in $\mathcal{M}^{\mathcal{T}}$; and
3. if $Q(x_1, \vec{y}) \rightsquigarrow A_1(x_1)$ is in $\mathcal{M}$ and $A_1 \sqsubseteq_{\mathcal{T}} A_2$, then $Q(x_1, \vec{y}) \rightsquigarrow A_2(x_1)$ is in $\mathcal{M}^{\mathcal{T}}$,

where we identify $P^-(y, x)$ and $P(x, y)$. □

The above definition of $\mathcal{T}$-mappings is designed so that the virtual ABoxes created by a $\mathcal{T}$-mapping will be H-complete. We state this as a theorem.

Theorem 4.3.2. *Let $\mathcal{O}$ be an ontology, $\mathcal{S}$ a database schema, $\mathcal{M}$ a mapping from $\mathcal{S}$ to $\mathcal{O}$, and $\mathcal{T}$ a TBox over $\mathcal{O}$. The virtual ABox $\mathcal{A}$ created by the $\mathcal{T}$-mapping $\mathcal{M}^{\mathcal{T}}$ is H-complete.*

Proof. The result follows directly from Definitions 3.5.3 and 4.3.1. We go through the details for point 2 of the definitions.

Suppose $A_2(a) \in \mathcal{A}$, and $A_2 \sqsubseteq_{\mathcal{T}} A_1$, we need to show that $A_1(a) \in \mathcal{A}$ for the second condition of Definition 3.5.3 to hold. Since $A_2(a) \in \mathcal{A}$, $\{x \mapsto a\}$ is a certain answer of $A_2(x)$, and there must be some mapping $m : Q(x, \vec{y}) \rightsquigarrow A_2(x)$, where $\{x \mapsto a\}$ is a certain answer to $Q(x)$. Since $A_2 \sqsubseteq_{\mathcal{T}} A_1$, by point 2 of Definition 4.3.1, there is a mapping assertion $m' : Q(x, \vec{y}) \rightsquigarrow A_1(x)$, so $\{x \mapsto a\}$ is also a certain answer to $A_1(x)$, and $A_1(a) \in \mathcal{A}$. The proofs for points 1 and 3 are analogous. □

Definition 4.3.3 (Simplified $\mathcal{T}$-mapping)**.** Let $\mathcal{M}^{\mathcal{T}}$ be a $\mathcal{T}$-mapping. If $\mathcal{M} \subseteq \mathcal{M}^{\mathcal{T}}$ is a $\mathcal{T}$-mapping, and for any conjunctive query **cq** the answers to the unfolding of **cq** over $\mathcal{M}$ are the same as the answers to the unfolding of **cq** over $\mathcal{M}^{\mathcal{T}}$, then $\mathcal{M}$ is a *simplified $\mathcal{T}$-mapping*. □

Note that by Definition 4.3.3, any $\mathcal{T}$-mapping is a simplification of itself. In the following, we will often refer to a simplified $\mathcal{T}$-mapping as, simply, a $\mathcal{T}$-mapping.

4.4 Simplifying the $\mathcal{T}$-mapping

The closer the ontology reflects the structure of the underlying data, the more redundancy we are likely to introduce into the mapping when composing it with the TBox. In Example 4.4.7, one concept is listed with three included concepts (subconcepts). These are then mapped to the same query as the main concept, but with filters specifying type.

[RKZ13] suggest using query containment and disjunctive filters as two ways of simplifying the $\mathcal{T}$-mapping. Although query containment checks can be prohibitively expensive, the $\mathcal{T}$-mapping only needs to be computed when either the mapping or the ontology changes, and so we can accept a larger cost of optimising it.

Definition 4.4.1 (Redundant mapping assertions)**.** Let $\mathcal{M} = \mathcal{M}' \cup \{m\}$ be a (simplified) $\mathcal{T}$-mapping where m is a mapping assertion. If $\mathcal{M}'$ is a simplified $\mathcal{T}$-mapping, then *m is a redundant mapping assertion in $\mathcal{M}$*. □

Before we are ready to study redundancies in the $\mathcal{T}$-mapping, we need to define query containment.

Definition 4.4.2 (Query containment)**.** Let $\mathcal{S}$ be a database schema, and Q_1 and Q_2 conjunctive queries over $\mathcal{S}$, where Q_1 and Q_2 have the same distinguished variables. We say that Q_1 *is contained in* Q_2, and write $Q_1 \subseteq Q_2$, if

$$Ans(Q_1, D) \subseteq Ans(Q_2, D),$$

or, equivalently,

$$CertAns(Q_1, D) \subseteq CertAns(Q_2, D),$$

for all instances D of $\mathcal{S}$. □

Using query containment, we can now check when one mapping assertion is redundant in the presence of another.

Theorem 4.4.3. *Let $\mathcal{M}^{\mathcal{T}}$ be a (simplified) $\mathcal{T}$-mapping, and $m : Q \rightsquigarrow \mathbf{q}$ and $m'_i : Q'_i \rightsquigarrow \mathbf{q}'_i$ mapping assertions in $\mathcal{M}^{\mathcal{T}}$. m is redundant in $\mathcal{M}^{\mathcal{T}}$ if and only if there is a collection of substitutions σ_i, such that $\mathbf{q} = \mathbf{q}'_i\sigma_i$ for each i, and $Q \subseteq \bigvee_i Q'_i\sigma_i$.*

Proof. For the "if" part, let $\mathcal{M}^{\mathcal{T}}$ be a $\mathcal{T}$-mapping, $m : Q \rightsquigarrow \mathbf{q}$ and $m'_1 : Q'_1 \rightsquigarrow \mathbf{q}'_i$ mapping assertions in $\mathcal{M}^{\mathcal{T}}$, and σ_i substitutions such that $\mathbf{q} = \mathbf{q}'_i\sigma_i$ and $Q \subseteq \bigvee_i Q'_i\sigma_i$. Let $\mathbf{cq}$ be a conjunctive query with an atom $\mathbf{p}$ that unifies with $\mathbf{q}$, and let CQ_m be some mapping rewriting of $\mathbf{cq}$ and $\mathcal{M}^{\mathcal{T}}$ based on M and μ, where μ is some unifier and $m \in M$ (see Definition 4.2.2 for the difference between mapping rewritings and the complete unfolding). Let $M^- = M \setminus \{m\}$. We have

$$CQ_m = \left(\left[\bigwedge_{m_j \in M^-} Q_j \right] \wedge Q \right) \mu.$$

Since $\mathbf{q} = \mathbf{q}'_i\sigma_i$, we know that $\mathbf{q}'_i$ can be made to match $\mathbf{p}$ using the composed substitution $\sigma_i\mu$. Thus the unfolding of the conjunctive query $\mathbf{cq}$ will also contain mapping rewritings

$$CQ_{m'_i} = \left(\left[\bigwedge_{m_j \in M^-} Q_j \right] \wedge Q'_i\sigma_i \right) \mu,$$

since $Q \subseteq \bigvee_i Q'_i\sigma_i$, it follows that $CQ_m \subseteq \bigvee_i CQ_{m'_i}$.

Since the only new mapping rewritings of $\mathbf{cq}$ and $\mathcal{M}$ we get by adding m to $\mathcal{M}$, are mapping rewritings that are contained in (combinations of) other mapping rewritings, $\mathcal{M}$ is a simplified $\mathcal{T}$-mapping if $\mathcal{M} \cup \{m\}$ is a (simplified) $\mathcal{T}$-mapping.

For the "only if" part, suppose there is no collection of mapping assertions $m'_i : Q'_i \rightsquigarrow \mathbf{q}'_i$ and substitutions σ_i, such that $\mathbf{q} = \mathbf{q}'_i\sigma_i$ and $Q \subseteq \bigvee_i Q'_i\sigma_i$. Now the mapping $m : Q \rightsquigarrow \mathbf{q}$ cannot be redundant. To see this, we look at the unfolding of $\mathbf{q}$. With m in the mapping, our unfolding contains the query Q. Since Q is not contained in any combination of other queries that $\mathbf{q}$ can be rewritten to, removing m from the mapping will remove answers from the unfolding.

We do not look at mapping assertions whose heads can be unified with $\mathbf{q}$ (instead of being matched to $\mathbf{q}$), because the only new mapping rewritings this would produce would be mapping rewritings where we have put equality restrictions or constant value restrictions on the answer variables of $\mathbf{q}$, and these rewritings cannot in general give the same answers as Q. □

See [LMR+14] for a slightly different approach to mapping redundancy.

Example 4.4.4. We look at a simple knowledge base $\mathcal{K}$ with TBox

$$\mathcal{T} = \{Manager \sqsubseteq Employee\}.$$

We define the mapping $\mathcal{M}$ with mapping assertions

$$\begin{aligned}
m_1 &: \texttt{SELECT id FROM employees} &&\rightsquigarrow Employee(id)\\
m_2 &: \texttt{SELECT id FROM employees}\\
&\quad\ \texttt{WHERE type = 1} &&\rightsquigarrow Manager(id).
\end{aligned}$$

We compose $\mathcal{M}$ and $\mathcal{T}$ to get the mapping $\mathcal{M}^{\mathcal{T}}$ with mapping assertions

$$\begin{aligned}
m_1 &: \texttt{SELECT id FROM employees} &&\rightsquigarrow Employee(id)\\
m_2 &: \texttt{SELECT id FROM employees}\\
&\quad\ \texttt{WHERE type = 1} &&\rightsquigarrow Manager(id)\\
m_3 &: \texttt{SELECT id FROM employees}\\
&\quad\ \texttt{WHERE type = 1} &&\rightsquigarrow Employee(id),
\end{aligned}$$

but since m_3 is redundant (in the presence of m_1), we can remove m_3. Thus the original mapping $\mathcal{M}$ was in fact a $\mathcal{T}$-mapping. This is because the only ontology axiom was mirrored by the structure of the underlying data. □

Example 4.4.4 deals with a very small TBox and a very small mapping. With large TBoxes and mappings, the $\mathcal{T}$-mapping can contain very many redundant mapping assertions. In order to avoid checking containment between two redundant mapping assertions (both of which will be removed), we should always check for containment against existing assertions when we try to add a new one to the $\mathcal{T}$-mapping (instead of adding first and optimising last).

Definition 4.4.5 (Queries with filters)**.** Let $Q(x, \vec{y})$ be a database query, then $Q(a, \vec{y})$ is shorthand notation for the query $Q(x, \vec{y}) \wedge (x = a)$, where $(x = a)$ is a *filter*. Depending on the database language, we may use more complex filters, for example disjunctions of value constrains. Typical value constraints are $=$, $<$, and $>$ for numbers, and similar constraints for other data types. □

Query containment due to filters are one of the cheapest kinds of query containment to look for.

Theorem 4.4.6. *Let $\mathcal{M} \cup \mathcal{M}_Q$ be a (simplified) $\mathcal{T}$-mapping, where*

$$\mathcal{M}_Q = \{m_i : Q(x, \vec{y}) \wedge (x = a_i) \rightsquigarrow \mathbf{q} \mid i = 1, \ldots, n\},$$

and x is not a distinguished variable of $\mathbf{q}$. Then

$$\mathcal{M} \cup \{m : Q(x, \vec{y}) \wedge (x = a_1 \vee \ldots \vee x = a_n) \rightsquigarrow \mathbf{q}\}$$

is a simplified $\mathcal{T}$-mapping.

Proof. That we can add m to $\mathcal{M} \cup \mathcal{M}_Q$ follows from the fact that a mapping rewriting using m cannot provide answers that are not provided by one of the mapping rewritings using one of m_i. Suppose σ is a certain answer to a mapping rewriting CQ_m using m. σ must assign x to a_i for some i. Thus, σ will also be a certain answer to the mapping rewriting CQ_{m_i}, where we use m_i in place of m.

In the presence of m, the assertions in $\mathcal{M}_Q$ plainly satisfy the redundancy conditions of Theorem 4.4.3, and so $\mathcal{M} \cup \{m\}$ is a simplified $\mathcal{T}$-mapping. □

Note that the body of the new mapping m in Theorem 4.4.6, is not a conjunctive query. This will not be a problem for us, since modern database management systems are very good at dealing with filters [RKZ13].

Example 4.4.7. We revisit Example 4.4.4, but we extend the TBox:

$$\begin{array}{lllll} \mathcal{T} = \{ & \mathit{Manager} & \sqsubseteq & \mathit{Employee} & \\ & \mathit{Clerk} & \sqsubseteq & \mathit{Employee} & \\ & \mathit{Engineer} & \sqsubseteq & \mathit{Employee} & \}. \end{array}$$

We define the mapping $\mathcal{M}$, with mapping assertions

$$\begin{array}{lll} m_1: \texttt{SELECT id FROM employees} & & \\ \quad \texttt{WHERE type = 1} & \rightsquigarrow & \mathit{Manager}(id) \\ m_2: \texttt{SELECT id FROM employees} & & \\ \quad \texttt{WHERE type = 2} & \rightsquigarrow & \mathit{Clerk}(id) \\ m_3: \texttt{SELECT id FROM employees} & & \\ \quad \texttt{WHERE type = 3} & \rightsquigarrow & \mathit{Engineer}(id). \end{array}$$

The $\mathcal{T}$-mapping becomes

$$\begin{array}{lll} m_1: \texttt{SELECT id FROM employees} & & \\ \quad \texttt{WHERE type = 1} & \rightsquigarrow & \mathit{Manager}(id) \\ m_2: \texttt{SELECT id FROM employees} & & \\ \quad \texttt{WHERE type = 2} & \rightsquigarrow & \mathit{Clerk}(id) \\ m_3: \texttt{SELECT id FROM employees} & & \\ \quad \texttt{WHERE type = 3} & \rightsquigarrow & \mathit{Engineer}(id) \\ m_4: \texttt{SELECT id FROM employees} & & \\ \quad \texttt{WHERE type = 1} & \rightsquigarrow & \mathit{Employee}(id) \\ m_5: \texttt{SELECT id FROM employees} & & \\ \quad \texttt{WHERE type = 2} & \rightsquigarrow & \mathit{Employee}(id) \\ m_6: \texttt{SELECT id FROM employees} & & \\ \quad \texttt{WHERE type = 3} & \rightsquigarrow & \mathit{Employee}(id). \end{array}$$

We use Theorem 4.4.6, to get

m_1 : `SELECT id FROM employees WHERE type = 1` $\rightsquigarrow$ $Manager(id)$

m_2 : `SELECT id FROM employees WHERE type = 2` $\rightsquigarrow$ $Clerk(id)$

m_3 : `SELECT id FROM employees WHERE type = 3` $\rightsquigarrow$ $Engineer(id)$

m_7 : `SELECT id FROM employees WHERE type = 1 OR type = 2 OR type = 3` $\rightsquigarrow$ $Employee(id)$,

which is the simplified $\mathcal{T}$-mapping $\mathcal{M}^{\mathcal{T}}$. □

4.5 $\mathcal{T}$-mapping optimisation versus ontology rewriting

Without H-complete ABoxes, we must expand our queries so that they reflect all the inclusion axioms of the TBox. The queries we add are exactly the queries we take into account when we generate the $\mathcal{T}$-mapping.

Example 4.5.1. We revisit the ontology and mapping of Example 4.4.7. We want to find the ontology rewriting of $\mathbf{q}(id) = Employee(id)$ (see [PLC+08] for ontology rewriting and unfolding algorithms). From the TBox inclusion axioms, we get the ontology rewriting

$$\mathbf{q}_0(id) = Employee(id) \vee Manager(id) \vee Clerk(id) \vee Engineer(id).$$

We then unfold this query into the database query

```
SELECT id FROM employees WHERE type = 1
   UNION
SELECT id FROM employees WHERE type = 2
   UNION
SELECT id FROM employees WHERE type = 3.
```

Before executing this query, we optimise it

```
SELECT id FROM employees WHERE type = 1
   OR type = 2 OR type = 3.
```

The final query is the same as we would get if we unfolded $\mathbf{q}$ over the $\mathcal{T}$-mapping of Example 4.4.7. □

Example 4.5.1 shows that we end up with the same query with and without the $\mathcal{T}$-mapping. Furthermore, we end up doing the same optimisation (Theorem 4.4.6).

Again, the advantage of using the $\mathcal{T}$-mapping is not in the potential optimisation of the end-query, but that much of this optimisation can be done once, in a way that benefits all subsequent rewritings.

The fact that the $\mathcal{T}$-mapping improvements can be done once (or at most whenever the mapping or ontology changes), means that we can make more expensive improvements. We can even consider full query containment checks.

4.6 Perfect rewritings and perfect mappings

In [PLL+13], Pinto et al. introduce the notions of perfect rewritings and perfect mappings.

Definition 4.6.1 (Perfect rewriting)**.** Let $\mathcal{B} = \langle \mathcal{O}, \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$ be an OBDA specification, $\mathbf{q}$ a query over $\mathcal{B}$, and Q a query over $\mathcal{S}$. Q is a *perfect rewriting* of $\mathbf{q}$ under $\mathcal{B}$ if

$$CertAns(\mathbf{q}, \mathcal{B}, D) = CertAns(Q, D)$$

for every instance D of $\mathcal{S}$. □

A perfect rewriting of an ontology query is a database query whose answers are exactly the certain answers of the ontology query. For example, any full rewriting and unfolding of a query is a perfect rewriting of that query.

Definition 4.6.2 (Perfect mapping)**.** Let $\mathcal{B} = \langle \mathcal{O}, \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$ be an OBDA specification, $\mathbf{q}$ a query over $\mathcal{B}$, and Q a query over $\mathcal{S}$ such that Q is a perfect rewriting of $\mathbf{q}$ under Q. Then

$$m : Q \rightsquigarrow \mathbf{q}$$

is a *perfect mapping assertion for* $\mathcal{B}$. A set of perfect mapping assertions for $\mathcal{B}$ is a *perfect mapping for* $\mathcal{B}$. □

We do not limit our perfect mappings to GAV mapping assertions. In fact, perfect mapping assertions can take the form of more general GLAV (existentially quantified variables in head as well as body) mapping assertions [PLL+13]. Still, we will for the most part limit our discussion to GAV mapping assertions.

Perfect GAV mapping assertions can also be used as regular GAV mapping assertions. If the head of the perfect mapping assertion is a conjunctive query, we simply create one mapping assertion for each conjunct. There is no guarantee that the new mapping assertions are perfect.

When applying perfect mappings in Chapter 6, we will need a placeholder for the subqueries that will be replaced using a perfect mapping.

Definition 4.6.3 (Split mappings)**.** Let $\mathcal{M}$ be a perfect mapping. We create the *high-level mapping* $\mathcal{M}^H$ and the *low-level mapping* $\mathcal{M}^L$ as follows.

For every $m_i : Q_i(\vec{x}, \vec{y}) \rightsquigarrow \mathbf{q}_i(\vec{x}, \vec{z})$ in $\mathcal{M}$, introduce a view predicate $v_i(\vec{x})$. Then, add $m_i^L : Q_i(\vec{x}, \vec{y}) \rightsquigarrow v_i(\vec{x})$ to $\mathcal{M}^L$, and $m_i^H : v_i(\vec{x}) \rightsquigarrow \mathbf{q}_i(\vec{x}, \vec{z})$ to $\mathcal{M}^H$. □

The view predicates in Definition 4.6.3 are allowed to serve as atomic ontology queries during the ontology rewriting described in Chapters 5 and 6, but since they are not proper names in the ontology language, they will not interfere with the rest of the rewriting (they do not occur in the TBox).

4.7 Creating a $\mathcal{T}$-mapping with perfect mappings

We now outline the steps to creating a $\mathcal{T}$-mapping. Some further simplification (step 3) regarding perfect mappings are discussed in Section 6.3.

Input: A mapping $\mathcal{M}$, a TBox $\mathcal{T}$, and a perfect mapping $\mathcal{M}_P$.

Output: A simplified $\mathcal{T}$-mapping $\mathcal{M}^{\mathcal{T}}$ with perfect mapping optimization.

Step 1: Make the mapping H-complete

Add new mapping assertions to $\mathcal{M}$ in order to make it H-complete (Definition 4.3.1).

Step 2: Add perfect mapping assertions

Add to $\mathcal{M}$ all the assertions of $\mathcal{M}_P$ and $\mathcal{M}_P^L$ (Definitions 4.6.2 and 4.6.3).

Step 3: Simplify

Simplify the mappings using disjunctive filters and query containment checks (Theorems 4.4.3 and 4.4.6 and Section 6.3). The resulting query is the simplified $\mathcal{T}$-mapping $\mathcal{M}^{\mathcal{T}}$.

□

The $\mathcal{T}$-mapping allows for simpler ontology rewritings, since it makes many of the TBox axioms redundant. In the next chapter, we look at the tree-witness ontology rewriting. We will then make use of the H-completeness of the virtual ABox created by a $\mathcal{T}$-mapping.

Chapter 5

Tree-witness ontology rewriting

In Chapter 3, we defined ontology rewritings: rewritings that let queries absorb TBoxes. In Chapter 4, we described a way to create H-complete virtual ABoxes by modifying the mapping. In this chapter, we present a simple version of the tree-witness rewriting. A version that is an ontology rewriting over H-complete ABoxes.

The tree-witness ontology rewriting [KLT+10, KKZ12, KKPZ13] (or just tree-witness rewriting) is an ontology rewriting (Definition 3.7.1). It uses the canonical model of an ontology to provide anonymous individuals (tree-witnesses) that can take the place of non-distinguished variables in a query.

Example 5.0.1. We look at the query $\mathbf{q}(x) = A(x) \land R(x,y)$. Since y is not a distinguished variable, we do not need to find an actual ABox individual to map it to, so long as we know there is some y satisfying $R(x,y)$ for our selected x.

Assume we have a single TBox axiom $A \sqsubseteq \exists R$, mapping assertions $m_1 : Q_A(x_1) \rightsquigarrow A(x_1)$ and $m_2 : Q_R(x_2,x_3) \rightsquigarrow R(x_2,x_3)$. In Definitions 3.5.3 and 4.3.1 (H-completeness) we modify the ABox (or the mapping) so it reflects the TBox axioms. We could not do this for axioms of the form $A \sqsubseteq \exists R$. If $A(x)$, then we know that $R(x,w)$ for some w, but we do not know which w.

Instead of modifying the mapping, we rewrite the query to

$$\mathbf{q}_0(x) = Q_A(x) \lor \exists y[Q_A(x) \land Q_R(x,y)].$$

The first disjunct of $\mathbf{q}_0$ reflects that our TBox axioms guarantee the existence of some w such that $R(x,w)$. $\mathbf{q}_0(x)$ is an (unfolded) ontology

rewriting of $\mathbf{q}(x)$ (Definition 3.7.1). We can simplify $\mathbf{q}_0(x)$ by dropping the second disjunct. □

5.1 The canonical model of a knowledge base

In order to find the certain answers to a query, we must prove that some assertions (concept and role) hold in every model of our knowledge base. Instead of constructing such proofs, we try to construct a canonical model for the knowledge base. The assertions that are true in the canonical model for a knowledge base, are exactly the assertions that are true in every model for that knowledge base. Using the canonical model, we can reduce proving validity to checking satisfaction in a model.

Definition 5.1.1 (Canonical models)**.** Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a knowledge base, and $\mathcal{C}_\mathcal{K}$ an interpretation of $\mathcal{K}$. $\mathcal{C}_\mathcal{K}$ is a *canonical model for* $\mathcal{K}$, if

$$\mathcal{C}_\mathcal{K} \models \Phi\sigma \qquad \text{iff} \qquad \mathcal{K} \models \Phi\sigma_{\mathrm{dvar}(\Phi)},$$

for any conjunctive query $\Phi = \exists \vec{y}\phi(\vec{x}, \vec{y})$, and any substitution σ mapping the variables in $\mathrm{dvar}(\Phi)$ into $\mathrm{ind}(\mathcal{A})$. □

There is no a priori reason to assume that every knowledge base has a canonical model. It does, however, turn out that every *consistent* knowledge base does have a canonical model [KKZ12].

Algorithm 5.1.2 (Canonical models)**.** This algorithm is from [KKZ12]. Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an OWL 2 QL knowledge base. For every role R with $\exists R$ occurring in $\mathcal{T}$, we introduce a new symbol w_R, called the witness for $\exists R$. We write $\mathcal{K} \models B(w_R)$ if $\exists R^- \sqsubseteq_\mathcal{T} B$.

We define a generating relation $\rightsquigarrow$ on the union of $\mathrm{ind}(\mathcal{A})$ and the set of witnesses as follows:

1. If $a \in \mathrm{ind}(\mathcal{A})$, R is $\sqsubseteq_\mathcal{T}$ minimal such that $\mathcal{K} \models \exists R(a)$, and there is no $b \in \mathrm{ind}(\mathcal{A})$ such that $\mathcal{K} \models R(a, b)$, then $a \rightsquigarrow w_R$.
2. If $u \rightsquigarrow w_{R_1}$ for some u, R_2 is $\sqsubseteq_\mathcal{T}$ minimal such that $\mathcal{K} \models \exists R_2(w_{R_1})$, and it is not the case that $R_1 \sqsubseteq_\mathcal{T} R_2^-$, then $w_{R_1} \rightsquigarrow w_{R_2}$.

If $a \rightsquigarrow w_{R_1} \rightsquigarrow \cdots \rightsquigarrow w_{R_n}$ for $n \geq 0$, we say that a *generates* the path $aw_{R_1} \cdots w_{R_n}$. Let $\mathrm{path}_\mathcal{K}(a)$ be the set of paths generated by a, and $\mathrm{tail}(\pi)$

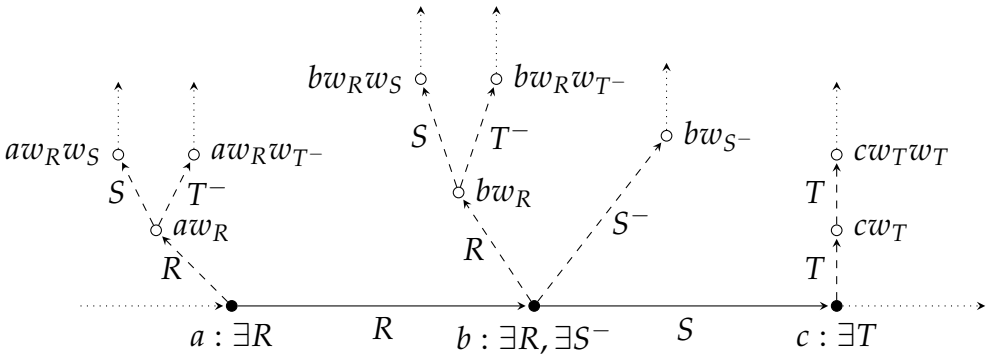

Figure 5.1: Multigraph representation of a canonical model. The black dots and solid arrows represent ABox assertions. The white dots and dashed arrows represent TBox induced facts. We have omitted concept assertions about a, b, and c, and the induced concept assertions about the witnesses. The three second level witnesses reflect that $\exists S \sqsubseteq_{\mathcal{T}} \exists R^-$, $\exists T^- \sqsubseteq_{\mathcal{T}} \exists R^-$, and $\exists T \sqsubseteq_{\mathcal{T}} \exists T^-$ (the domain of the second role is contained in the range of the first).

be the last element in the path π. The path $\pi \cdot x$ is the path obtained by adding x to the end of π.

We define the canonical model $\mathcal{C}_\mathcal{K}$ by:

$$
\begin{aligned}
\Delta^{\mathcal{C}_\mathcal{K}} &= \bigcup_{a \in \mathrm{ind}(\mathcal{A})} \mathrm{path}_\mathcal{K}(a), \\
a^{\mathcal{C}_\mathcal{K}} &= a \qquad \text{for all } a \in \mathrm{ind}(\mathcal{A}), \\
A^{\mathcal{C}_\mathcal{K}} &= \{\pi \in \Delta^{\mathcal{C}_\mathcal{K}} \mid \mathcal{K} \models A(\mathrm{tail}(\pi))\}, \\
P^{\mathcal{C}_\mathcal{K}} &= \{\langle a, b\rangle \in \mathrm{ind}(\mathcal{A}) \times \mathrm{ind}(\mathcal{A}) \mid \mathcal{K} \models P(a,b)\} \\
&\quad \cup \{\langle \pi, \pi \cdot w_R\rangle \mid \mathrm{tail}(\pi) \rightsquigarrow w_R \text{ and } R \sqsubseteq_\mathcal{T} P\} \\
&\quad \cup \{\langle \pi \cdot w_R, \pi\rangle \mid \mathrm{tail}(\pi) \rightsquigarrow w_R \text{ and } R \sqsubseteq_\mathcal{T} P^-\}.
\end{aligned}
$$

The set $\Delta^{\mathcal{C}_\mathcal{K}} \setminus \mathrm{ind}(\mathcal{A})$ is called the *tree part* of $\mathcal{C}_\mathcal{K}$. □

Note that there are queries where $\mathcal{C}_\mathcal{K} \models \Phi\sigma$, even though $\mathcal{K} \not\models \Phi\sigma$, but only if we allow σ to map answer variables to individuals outside $\mathrm{ind}(\mathcal{A})$. Different models may satisfy the TBox in different ways ($\mathcal{C}_\mathcal{K}$ uses the tree-witnesses), and as such they need not agree on all answers to all queries.

Figure 5.1 shows a multigraph representation of the canonical model for some knowledge base, as generated by Algorithm 5.1.2. The graph consists of a multigraph representation of the ABox, where every node is the root of a tree of witnesses (paths in the canonical model). Since we use a virtual ABox, we must query the database for the ABox part of the canonical model. The only parts of the canonical model we can compute from our knowledge base, are the trees of witnesses.

Definition 5.1.3 (Restricted canonical model)**.** Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a knowledge base on normal form. A *canonical model for $\mathcal{K}$ restricted to R and a* (or a *restricted canonical model*), written $\mathcal{C}^R_{\mathcal{K}}(a)$, is a canonical model for $\mathcal{K} = \langle \mathcal{T}, \{A_R(a)\} \rangle$. Recall that $A_R \equiv \exists R$ is a TBox axiom for knowledge bases on normal form. □

In Figure 5.1, $\mathcal{C}^R_{\mathcal{K}}(a)$ is the dotted arrow tree with a at its root. $\mathcal{C}^{S^-}_{\mathcal{K}}(b)$ is the right-hand part of the dotted arrow tree with b at its root.

When working with query answering in the canonical model of a knowledge base, it is useful to have dedicated concepts for the domains and ranges of assertions. The following definition provides such concepts.

Definition 5.1.4 (Knowledge base normal form)**.** Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a knowledge base. For every role R in $\mathcal{T}$, create new concepts A_R (the domain of R) and A_{R^-} (the range of R), and add to $\mathcal{T}$ the axioms

$$A_R \equiv \exists R, \qquad A_{R^-} \equiv \exists R^-.$$

The new $\mathcal{K}$ is on *normal form*. (The concepts A_R and A_{R^-} can not be used in the ABox or other TBox axioms; they must be new.) □

5.2 Query answering in the canonical model

According to Definition 5.1.1, we can find the certain answers to a query over a knowledge base, using only the canonical model. The canonical model reflects the ABox assertions in the knowledge base, and adds new symbols to play the role of anonymous witnesses whenever such a witness is guaranteed to exist. By restricting the range of the answers (answer substitutions) to $\mathrm{ind}(\mathcal{A})$, we guarantee that no arbitrary facts about anonymous witnesses will be reflected in a query answer.

5.2.1 Multigraph representation of queries

A query can be represented as a multigraph, where the variables are nodes and the role assertions are edges. Concept assertions are then stored as information about the nodes.

Definition 5.2.1 (Multigraph representation of queries)**.** A *multigraph representation of a query* $\mathbf{q}$ is a directed graph $\langle V_{\mathbf{q}}, E_{\mathbf{q}} \rangle$ and a labelling function $f_{\mathbf{q}}$, where $V_{\mathbf{q}}$ (the vertices) is a set containing the variables of $\mathbf{q}$, and $E_{\mathbf{q}}$ (the edges) is a set containing the pairs of variables connected by role assertions in $\mathbf{q}$. For every concept A, $A \in f_{\mathbf{q}}(v)$ if and only if $A(v) \in \mathbf{q}$. For every role R, $R \in f_{\mathbf{q}}(\langle x, y \rangle)$ if and only if $R(x, y) \in \mathbf{q}$.

As usual, $\mathrm{dvar}(\mathbf{q})$ is the set of distinguished (answer) variables in $\mathbf{q}$, $\mathrm{nvar}(\mathbf{q})$ is the set of non-distinguished variables in $\mathbf{q}$. □

The graph in Definition 5.2.1 is not itself a multigraph, but when $f(e)$ contains more than one role, the original query had more than one role connecting the variables of e. Together, the graph and the labelling function can produce a labelled multigraph.

In order to check if a query holds in the canonical model, we need a way to check if a query can be matched to the canonical model. For this, we need a restricted class of substitutions, called *homomorphisms*.

Definition 5.2.2 (Homomorphism)**.** Let $\mathbf{cq}_1$ and $\mathbf{cq}_2$ be conjunctive queries, and σ a substitution. If

1. σ maps the non-distinguished (existentially quantified) variables of $\mathbf{cq}_1$ to the non-distinguished variables of $\mathbf{cq}_2$,
2. σ maps the distinguished (free, or answer) variables of $\mathbf{cq}_1$ to the distinguished variables of $\mathbf{cq}_2$ and constants, and
3. $\mathbf{cq}_1\sigma$ is a subformula of $\mathbf{cq}_2$,

then σ is a *homomorphism from* $\mathbf{cq}_1$ *to* $\mathbf{cq}_2$. □

Using the multigraph representation, we reduce answering a query to finding graph homomorphisms from the nodes of the query graph to the nodes of the graph of the canonical model (Figure 5.1). When a query (with all variables replaced by individual names) matches the canonical model, each atom of that query is satisfied in the canonical model. Then, the used assignment of individual names to answer variables is a

certain answer to the query. We will state the connection between query answering and multigraph matching as Theorem 5.2.4.

The fact that we require answer variables to be mapped to the ABox part of the canonical model, lets us throw away parts of the canonical model. Since the queries are finite, there is a bound on how deep we need to inspect the (possibly infinite) witness trees. In the query graph, we can calculate the largest distance from any non-distinguished variable to its closest distinguished variable. Since the distinguished variables cannot be mapped to a witness tree node, the calculated length is an upper bound on the required depth of witness trees.

5.2.2 Multigraph representation of the canonical model

We build the multigraph representation of the canonical model based on the algorithm for producing canonical models (Algorithm 5.1.2).

Definition 5.2.3 (Multigraph representation of the canonical model)**.** Let $\mathcal{C}_\mathcal{K}$ be a canonical model for $\mathcal{K}$. The *multigraph representation of* $\mathcal{C}_\mathcal{K}$ is a graph $\langle V_{\mathcal{C}_\mathcal{K}}, E_{\mathcal{C}_\mathcal{K}} \rangle$ and a labelling function $f_{\mathcal{C}_\mathcal{K}}$, where $V_{\mathcal{C}_\mathcal{K}} = \Delta^{\mathcal{C}_\mathcal{K}}$, and $E_{\mathcal{C}_\mathcal{K}} = \bigcup_P P^{\mathcal{C}_\mathcal{K}}$. ($\Delta^{\mathcal{C}_\mathcal{K}}$ and $P^{\mathcal{C}_\mathcal{K}}$ are defined in Algorithm 5.1.2.) For every concept A, $A \in f_{\mathcal{C}_\mathcal{K}}(v)$ if and only if $v \in A^{\mathcal{C}_\mathcal{K}}$. For every role R, $R \in f_{\mathcal{C}_\mathcal{K}}(e)$ if and only if $e \in R^{\mathcal{C}_\mathcal{K}}$.

We define the witness part of $E_{\mathcal{C}_\mathcal{K}}$ as $\mathrm{wit}(\mathcal{K}) = E_{\mathcal{C}_\mathcal{K}} \setminus \mathrm{ind}(\mathcal{A})$. □

We can now answer queries by matching the multigraph representation of the query to the multigraph representation of the canonical model.

Theorem 5.2.4. *Let* $\mathbf{q}$ *be a query over* $\mathcal{K}$*, and* $\mathcal{C}_\mathcal{K}$ *a canonical model for* $\mathcal{K}$*. Let* σ *be a substitution mapping* $dvar(\mathbf{q})$ *to* $ind(\mathcal{A})$*, and* $nvar(\mathbf{q})$ *to* $wit(\mathcal{K}) \cup ind(\mathcal{A})$*.* $\sigma_{dvar(\mathbf{q})} \in CertAns(\mathbf{q}, \mathcal{K})$ *if and only if the multigraph representation of* $\mathbf{q}\sigma$ *is a subgraph of the multigraph representation of the canonical model, such that* $f_\mathbf{q} \subseteq f_{\mathcal{C}_\mathcal{K}} \restriction (E_\mathbf{q} \cup V_\mathbf{q})$*.*

Proof. The certain answers to a query over a knowledge base are exactly the answers we get from the canonical model (Definition 5.1.1). What we must show is that $\mathcal{C}_\mathcal{K} \models \mathbf{q}\sigma$ if and only if $\mathbf{q}\sigma$ is a subgraph of the multigraph representation of the canonical model, such that $f_{\mathbf{q}\sigma} \subseteq f_{\mathcal{C}_\mathcal{K}} \restriction (E_{\mathbf{q}\sigma} \cup V_{\mathbf{q}\sigma})$.

Assume $\mathcal{C}_\mathcal{K} \models \mathbf{q}\sigma$. It is straightforward to check that $E_{\mathbf{q}\sigma} \subseteq E_{\mathcal{C}_\mathcal{K}}$ and $V_{\mathbf{q}\sigma} \subseteq V_{\mathcal{C}_\mathcal{K}}$. Note that if $\mathcal{C}_\mathcal{K} \models \mathbf{q}\sigma$, then every atom in $\mathbf{q}\sigma$ is true in $\mathcal{C}_\mathcal{K}$.

Let $A \in f_{\mathbf{q}\sigma}(x)$. Then $A(x)$ is a concept atom in $\mathbf{q}\sigma$, so $\mathcal{C}_\mathcal{K} \models A(x)$. That means $x \in A^{\mathcal{C}_\mathcal{K}}$, and by definition of $f_{\mathcal{C}_\mathcal{K}}$, $A \in f_{\mathcal{C}_\mathcal{K}}(x)$.

Now assume $E_{\mathbf{q}\sigma} \subseteq E_{\mathcal{C}_\mathcal{K}}$, $V_{\mathbf{q}\sigma} \subseteq V_{\mathcal{C}_\mathcal{K}}$ and $f_{\mathbf{q}\sigma} \subseteq f_{\mathcal{C}_\mathcal{K}} \restriction (E_{\mathbf{q}\sigma} \cup V_{\mathbf{q}\sigma})$. We need to show that every atom of $\mathbf{q}\sigma$ is true in $\mathcal{C}_\mathcal{K}$. Let $A(x)$ be a concept atom in $\mathbf{q}\sigma$. Then $x \in E_{\mathbf{q}\sigma}$ and $A \in F_{\mathbf{q}\sigma}(x)$. Then, by our assumptions, $x \in E_{\mathcal{C}_\mathcal{K}}$ and $A \in F_{\mathcal{C}_\mathcal{K}}(x)$, so $x \in A^{\mathcal{C}_\mathcal{K}}$, and $\mathcal{C}_\mathcal{K} \models A(x)$.

The proofs for role atoms are essentially the same. □

5.2.3 Complexity of multigraph matching

In Theorem 5.2.4, we match a query to the canonical model after we have applied a substitution to the query. In order to find the appropriate substitution, we look for homomorphisms between $\langle E_\mathbf{q}, V_\mathbf{q}\rangle$ and $\langle E_{\mathcal{C}_\mathcal{K}}, V_{\mathcal{C}_\mathcal{K}}\rangle$. We limit our search to homomorphisms mapping $\mathrm{dvar}(\mathbf{q})$ to $\mathrm{ind}(\mathcal{A})$.

The problem of matching the multigraph representation of a query to the multigraph representation of the canonical model is NP-hard. The proof of this is by reduction of the 3-coloring problem for undirected graphs.

Theorem 5.2.5. *Matching the multigraph representation of a query to the multigraph representation of a canonical model is NP-hard.*

Proof. Let $\mathcal{A} = \{E(v_1, v_2), E(v_2, v_3), E(v_3, v_1)\}$ be an ABox and $\mathcal{K} = \langle \emptyset, \mathcal{A}\rangle$ a knowledge base. The multigraph representation of $\mathcal{C}_\mathcal{K}$ is now the graph with vertices $V_{\mathcal{C}_\mathcal{K}} = \{v_1, v_2, v_3\}$, and edges $E_{\mathcal{C}_\mathcal{K}} = \{\langle v_1, v_2\rangle, \langle v_2, v_3\rangle, \langle v_3, v_1\rangle\}$. $f_{\mathcal{C}_\mathcal{K}}(e_i) = \{E\}$ and $f_{\mathcal{C}_\mathcal{K}}(v_i) = \emptyset$. A graph G is 3-colorable if there is a homomorphism from G to $\langle E_{\mathcal{C}_\mathcal{K}}, V_{\mathcal{C}_\mathcal{K}}\rangle$. Given G, we construct the query $\mathbf{q}$ such that $G = \langle E_\mathbf{q}, V_\mathbf{q}\rangle$ and $f_\mathbf{q}(e_i) = \{E\}$ and $f_\mathbf{q}(v_i) = \emptyset$. Now G is 3-colorable if $\mathcal{C}_\mathcal{K} \models \mathbf{q}$. □

The above argument can also be used to show that the general problem of finding homomorphisms between graphs (or queries) is NP-hard.

When analysing the complexity of matching queries to the canonical model for a knowledge base with a virtual ABox, we cannot use the proof of Theorem 5.2.5. This is because we cannot create the required structure in the ABox (there will be no ABox part of the multigraph, and the trees cannot contain the required cycles).

5.3 The tree-witness rewriting

In the following, we will often need to treat a query as a set of atoms. Usually, a set of atoms is regarded as a conjunction. A set of such conjunctions is usually regarded as a disjunction. The distinction will be made explicit when it is not clear from the context.

Definition 5.3.1. Let $\Phi = \exists \vec{y} \phi(\vec{x}, \vec{y})$ be a conjunctive query, then $\mathbf{q}_\Phi(\vec{x})$ (or simply $\mathbf{q}(\vec{x})$) is the set of atoms in Φ, where $\vec{x}$ specifies the distinguished (non-quantified) variables of Φ.

When no confusion may arise we will use $\mathbf{q}(\vec{x})$ or $\mathbf{q}$ to refer to a query itself, as well as the set of atoms in the query. As usual, $\mathrm{nvar}(\mathbf{q})$ is the set of non-distinguished variables in $\mathbf{q}$, $\mathrm{dvar}(\mathbf{q})$ is the set of distinguished variables in $\mathbf{q}$, and $\mathrm{var}(\mathbf{q})$ is the set of all variables in $\mathbf{q}$. □

We are now ready to define the tree-witness, one of the key components to the tree-witness rewriting.

Definition 5.3.2 (Tree-witness)**.** Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a knowledge base on normal form, $\mathbf{q}(\vec{x})$ a conjunctive query over $\mathcal{K}$, and $R(x, y) \in \mathbf{q}$ an atom where $x \in \mathrm{dvar}(\mathbf{q})$ and $y \in \mathrm{nvar}(\mathbf{q})$. The pair $\mathfrak{t} = \langle \mathfrak{t}_r, \mathfrak{t}_w \rangle$ is a *tree-witness for* $\mathbf{q}$ *and* $\mathcal{T}$ *generated by* R if

$$\mathbf{q}_\mathfrak{t} = \{S(x, y) \in \mathbf{q} \mid x \in \mathfrak{t}_w \vee y \in \mathfrak{t}_w\} \cup \{S(x) \in \mathbf{q} \mid x \in \mathfrak{t}_w\},$$

is $\subseteq$-minimal such that

1. $x \in \mathfrak{t}_r$, $y \in \mathfrak{t}_w$, $\mathfrak{t}_r \subseteq \mathrm{var}(\mathbf{q})$, $\mathfrak{t}_w \subseteq \mathrm{nvar}(\mathbf{q})$, and $\mathfrak{t}_r \cap \mathfrak{t}_w = \emptyset$;
2. if $x \in \mathfrak{t}_w$ and either $R(x, y) \in \mathbf{q}$ or $R(y, x) \in \mathbf{q}$, then $y \in \mathfrak{t}_r \cup \mathfrak{t}_w$; and
3. there is a substitution σ mapping the variables in $\mathfrak{t}_r$ to some $a \in \mathrm{ind}(\mathcal{A})$, and the variables of $\mathfrak{t}_w$ to elements of $\Delta^{\mathcal{C}_\mathcal{K}} \setminus \mathrm{ind}(\mathcal{A})$, such that $\mathcal{C}^R_\mathcal{K}(a) \models \mathbf{q}_\mathfrak{t}\sigma$,

where $\mathcal{C}^R_\mathcal{K}(a)$ is a canonical model for $\mathcal{K}$ restricted to R and a.

If $\mathfrak{t}$ can be generated by more than one role, we let $\Omega_\mathfrak{t}$ denote the set of roles generating $\mathfrak{t}$. □

Tree-witnesses tell us what information we can infer based on the TBox. More precisely, a tree-witness tells us what atoms we may drop from

a query, based on the existence of other atoms (and possibly some constraints on remaining variables).

The query $\mathbf{q}_\mathfrak{t}$ in Definition 5.3.2 can be rewritten as (see [KKPZ13])

$$\exists z \left[A_R(z) \wedge \bigwedge_{x \in \mathfrak{t}_r} (x = z) \right].$$

Any z that satisfies the above formula will automatically satisfy all the dropped atoms of $\mathbf{q}_\mathfrak{t}$. The tree-witness $\mathfrak{t}$ maps all $x \in \mathfrak{t}_r$ to the root of some $\mathcal{C}^R_\mathcal{K}(a)$. Any individual in A_R is such a root. The variables $y \in \mathfrak{t}_w$ are mapped to witnesses in $\mathcal{C}^R_\mathcal{K}(a)$, and can safely be removed from the query.

The substitution σ (in point 3 of Definition 5.3.2) can be thought of as a homomorphism from $\mathbf{q}_\mathfrak{t}$ to $\mathcal{C}^R_\mathcal{T}(a)$, showing how the structure of the query matches (or can be collapsed to match) the restricted canonical model, thus guaranteeing the existence of the necessary witnesses for the non-distinguished variables.

Point 2 in Definition 5.3.2 is designed to make sure that every instance of a variable assigned to a witness occurs in the tree-witness. Otherwise this variable is likely to also be assigned to some other witness or even an individual in $\mathrm{ind}(\mathcal{A})$, and there is no guarantee that these two assignments will be to the same individual.

The requirement that $\mathbf{q}_\mathfrak{t}$ is minimal ensures that a tree-witness is not really a combination of two tree-witnesses. If one tree-witness could really be split into two tree-witnesses, then having two separate tree-witnesses instead of one large could allow more combinations of tree-witnesses (see Definition 5.3.7). Thus, keeping the combined tree-witness instead of the separate tree-witnesses may invalidate the rewriting.

Definition 5.3.3 (Tree-witness consistency)**.** Let $\mathfrak{t}^1$ and $\mathfrak{t}^2$ be two tree-witnesses. $\mathfrak{t}^1$ and $\mathfrak{t}^2$ are *consistent* if $\mathfrak{t}^1_w \cap \mathfrak{t}^2_w = \emptyset$. Θ is a consistent set (possibly empty or singleton) of tree-witnesses if the elements of Θ are pairwise consistent.

We can also express tree-witness consistency in terms of the queries $\mathbf{q}_\mathfrak{t}$.

Theorem 5.3.4. *Let $\mathfrak{t}^1$ and $\mathfrak{t}^2$ be two tree-witnesses. $\mathfrak{t}^1$ and $\mathfrak{t}^2$ are consistent if and only if $\mathbf{q}_{\mathfrak{t}^1} \cap \mathbf{q}_{\mathfrak{t}^2} = \emptyset$.*

Proof. Assume the tree-witnesses are not consistent. Then, by construction of the $\mathbf{q}_\mathfrak{t}$, the queries will not be disjoint (they share every atom containing the shared variable).

For the other direction, assume $\mathbf{q}_{\mathfrak{t}^1} \cap \mathbf{q}_{\mathfrak{t}^2} \neq \emptyset$. By definition, $\mathbf{q}_\mathfrak{t}$ contains only atoms with at least one variable from $\mathfrak{t}_w$, so the two tree-witnesses share at least one witness variable, and are inconsistent. □

If two tree-witnesses are inconsistent, they can not be used at the same time, because they may require assignments of different witnesses to the same variables. It is important to note that the same witness name need not refer to the same actual witness in two different tree-witnesses.

Definition 5.3.5 (Tree-witness query)**.** Let $\mathfrak{t}$ be a tree-witness. The *tree-witness query for* $\mathfrak{t}$ is the query

$$\mathbf{tw}_\mathfrak{t} = \bigvee_{R \in \Omega_\mathfrak{t}} \exists z \left[A_R(z) \wedge \bigwedge_{x \in \mathfrak{t}_r} (x = z) \right],$$

where $\Omega_\mathfrak{t}$ is the set of roles generating $\mathfrak{t}$, and $A_R \equiv \exists R$. □

The tree-witness query for tree-witness $\mathfrak{t}$ gives us all the assignments to the variables of $\mathfrak{t}_r$ that guarantee the existence of tree-witnesses for the variables of $\mathfrak{t}_w$.

Example 5.3.6. We have a TBox with one axiom

$$\exists S \sqsubseteq \exists R^-,$$

and a query

$$\mathbf{q}(x_1) = R(x_1, x_2) \wedge R(x_3, x_2) \wedge S(x_2, x_4).$$

The query has one tree-witness

$$\mathfrak{t} = \langle \{x_1, x_3\}, \{x_2, x_4\} \rangle$$

generated by R, and we get the tree-witness query

$$\mathbf{tw}_\mathfrak{t} = \exists z \big[A_R(z) \wedge (x_1 = z) \wedge (x_3 = z) \big].$$

The tree-witness query does not need to contain x_2 and x_4, because the tree-witness guarantees that there will be some witness matching them.

We could also try the tree-witness $\mathfrak{t}' = \langle\{x_1\}, \{x_2, x_3, x_4\}\rangle$, but the graph of $\mathbf{q}_{\mathfrak{t}'}$ could not be made to match $\mathcal{C}^R_{\mathcal{K}}(a)$ for our simple knowledge base. The problem would be with $R(x_3, x_2)$, since now, $\sigma(x_1) \neq \sigma(x_2)$. We have that $\sigma(x_1) = a$ is the root, and since $R(x_1, x_2)$, we have $\sigma(x_2) = aw_R$. Since $\exists S \sqsubseteq \exists R^-$, we know that $\mathcal{C}^R_{\mathcal{K}}(a) \models S(aw_R, aw_R w_S)$, so we let $\sigma(x_4) = aw_R w_S$. The only remaining variable is x_3, but since $aw_R w_{R^-}$ is not an individual in $\mathcal{C}_{\mathcal{K}}$, there is nothing σ can map x_3 to that would make $\mathcal{C}^R_{\mathcal{K}}(a) \models \mathbf{q}\sigma$. □

In Example 5.3.6, we say that $aw_R w_{R^-}$ is not an element of $\mathcal{C}_{\mathcal{K}}$ for our knowledge base. In fact, this element cannot be an element of any knowledge base, because there is no guarantee that there is such an element different from a (since $R \sqsubseteq (R^-)^-$, this element violates point 2 in Algorithm 5.1.2).

With all the tree-witnesses of a query, we can create the ontology rewriting called the tree-witness rewriting.

Definition 5.3.7 (Tree-witness rewriting)**.** Let $\mathbf{q}(\vec{x})$ be a conjunctive query over an H-complete knowledge base $\mathcal{K} = \langle\mathcal{T}, \mathcal{A}\rangle$ on normal form. The *tree-witness rewriting of* $\mathbf{q}$ *and* $\mathcal{T}$ is the query

$$\mathbf{q}_{\mathbf{tw}}(\vec{x}) = \bigvee_{\text{consistent } \Theta} \exists \vec{y} \left[(\mathbf{q} \setminus \mathbf{q}_\Theta) \wedge \bigwedge_{\mathfrak{t} \in \Theta} \mathbf{tw}_\mathfrak{t} \right],$$

where $\mathbf{q}_\Theta = \{\mathbf{q}_\mathfrak{t} \mid \mathfrak{t} \in \Theta\}$. □

We now prove that the tree-witness rewriting is in fact an ontology rewriting over H-complete ABoxes.

Theorem 5.3.8. *Let $\mathcal{K} = \langle\mathcal{T}, \mathcal{A}\rangle$ be a knowledge base with an H-complete ABox, and $\mathbf{q}(\vec{x})$ a conjunctive query over $\mathcal{K}$. If $\mathbf{q}(\vec{x})$ has at least one answer variable, the tree-witness rewriting of $\mathbf{q}(\vec{x})$ and $\mathcal{T}$ is an ontology rewriting of $\mathbf{q}(\vec{x})$ and $\mathcal{T}$ for H-complete ABoxes.*

Proof. The tree-witness rewriting is a disjunction of each way the original query may be split, and then mapped partially into the trees of the canonical model, and partially into the ABox part of the canonical model.

By point 3 of Definition 5.3.2, every tree-witness matches some restriction of the canonical model (some witness tree). Thus, the tree-witness rewriting is sound.

In order to show that the tree-witness rewriting is in fact a complete ontology rewriting, we need to show that every way of mapping a query into the canonical model can be represented by some collection of tree-witnesses, along with queries over the virtual ABox. We assume our query is a connected graph. Otherwise, we answer it as multiple separate queries (see the discussion following this proof).

Suppose $\mathbf{q}$ is mapped into $\mathcal{C}_\mathcal{K}$. The parts of $\mathbf{q}$ that are mapped into the ABox part of $\mathcal{C}_\mathcal{K}$ will be answered by regular queries. We must show that the parts mapped into the restrictions of $\mathcal{C}_\mathcal{K}$ (the witness trees) are in fact tree-witnesses. Let $\mathbf{p}$ be the subquery of $\mathbf{q}$ mapped into $\mathcal{C}^R_\mathcal{K}(a)$. We show that each of the conditions of Definition 5.3.2 holds.

If we let $\mathfrak{t}_r$ be the set of all variables of $\mathbf{p}$ mapped to the root of $\mathcal{C}^R_\mathcal{K}(a)$, and $\mathfrak{t}_w$ the remaining variables of $\mathbf{p}$, then condition 1 holds.

Since $\mathbf{q}$ is mapped into $\mathcal{C}_\mathcal{K}$ by a homomorphism μ, condition 2 holds. Otherwise, there would be some edge $R(x, y)$ in $\mathbf{p}$ connecting a witness x in $\mathcal{C}^R_\mathcal{K}(a)$ to some part y of $\mathcal{C}_\mathcal{K}$ not in $\mathcal{C}^R_\mathcal{K}(a)$. Then μ would not be a homomorphism from $\mathbf{q}$ to $\mathcal{C}_\mathcal{K}$.

Condition 3 holds by assumption. The only condition remaining is the minimality of the tree-witness. If minimality does not hold, we prove that $\mathbf{p}$ is a union of (disjoint) queries that each define a tree-witness (Example 5.3.9). Since $\mathbf{p}$ is not minimal, there is some subset $\mathbf{p}'$ of $\mathbf{p}$ for which is. Let $\mathbf{p}_0 = \mathbf{p} \setminus \mathbf{p}'$. $\mathbf{p}'$ now defines a tree-witness. We check if $\mathbf{p}_0$ also satisfies Definition 5.3.2. Since $\mathbf{p}$ satisfies condition 2, and $\mathbf{p}'$ and $\mathbf{p}_0$ are disjoint, $\mathbf{p}_0$ must also satisfy condition 2 (otherwise $\mathbf{p}$ would not).

Condition 3 still holds by assumption. We define $\mathfrak{t}$ as for $\mathbf{p}$. Some variable of $\mathbf{p}$, mapped to the root of $\mathcal{C}^R_\mathcal{K}(a)$, must also be in $\mathbf{q}_0$. Otherwise, the connection between $\mathbf{p}_0$ and the rest of the query would have to be through $\mathbf{p}'$, which it is not by definition of $\mathbf{p}'$, so condition 1 holds.

The only thing that remains to check is the minimality of $\mathbf{p}_0$. If this fails, repeat the construction of $\mathbf{p}'$ and $\mathbf{p}_0$, with $\mathbf{p}'_0$ the minimal subset of $\mathbf{p}_0$ satisfying Definition 5.3.2, and $\mathbf{p}_1 = \mathbf{p}_0 \setminus \mathbf{p}'_0$. Repeat until $\mathbf{p}_n$ is a tree-witness. □

If a query does not have any distinguished variables, we may map the entire query into some $\mathcal{C}^R_\mathcal{K}(a)$ (see [KKZ12] for an extension of the tree-witness rewriting to cover this case). We will assume that our queries always have at least one distinguished variable.

The answers to a disconnected query are based on the cross product of

the answers to the separate queries. Note that if two queries share a variable, they cannot be disconnected. A query without distinguished variables is a boolean query. If a boolean query forms a disconnected part of a query, then that query is a condition for there to be query answers.

Example 5.3.9. We assume the single TBox axiom $\exists S \sqsubseteq \exists R^-$ from Example 5.3.6, and look at a variation of the query from that Example (the second atom has new variables),

$$\mathbf{q}(x_1) = R(x_1, x_2) \wedge R(x_1, x_3) \wedge S(x_2, x_4).$$

We have been given the homomorphism

$$\mu = \{x_1 \mapsto a, \quad x_2 \mapsto aw_R, \quad x_3 \mapsto aw_R, \quad x_4 \mapsto aw_R w_S\}$$

from $\mathbf{q}$ into $\mathcal{C}_{\mathcal{K}}^R(a)$. This homomorphism forms the basis for the tree-witness $\mathfrak{t} = \langle\{x_1\}, \{x_2, x_3, x_4\}\rangle$, but this tree-witness is not minimal. Instead, we get the two tree-witnesses $\mathfrak{t}^1 = \langle\{x_1\}, \{x_2, x_4\}\rangle$ and $\mathfrak{t}^2 = \langle\{x_1\}, \{x_3\}\rangle$. □

5.4 Finding tree-witnesses

The multigraph representation of the canonical model is a good framework for understanding the tree-witness rewriting. Each tree-witness has a corresponding subquery $\mathbf{q}_\mathfrak{t}$ that matches a part of some witness tree in the canonical model multigraph. The substitution in Definition 5.3.2 is the homomorphism defining the match.

The tree-witness rewriting is based on the observation that, given a query $\mathbf{q}$ and a substitution σ, the multigraph representation of $\mathbf{q}\sigma$ matches the canonical model multigraph if parts of $\mathbf{q}\sigma$ match the witness trees, and the rest matches the ABox part. Point 2 in Definition 5.3.2 guarantees that a witness tree part of the graph of $\mathbf{q}\sigma$ is only connected to the rest of the graph at the root (which is in $\text{ind}(\mathcal{A})$); point 2 of Definition 5.3.2 requires that a tree-witness must contain every variable connected to its witness variables.

Example 5.4.1. Consider the query

$$\begin{aligned}\mathbf{q}(x_3, x_6) = \{&C(x_4), C(x_6), R(x_1, x_2), R(x_1, x_3),\\ &S(x_2, x_4), S(x_5, x_3), S(x_5, x_6), T(x_2, x_4)\}.\end{aligned}$$

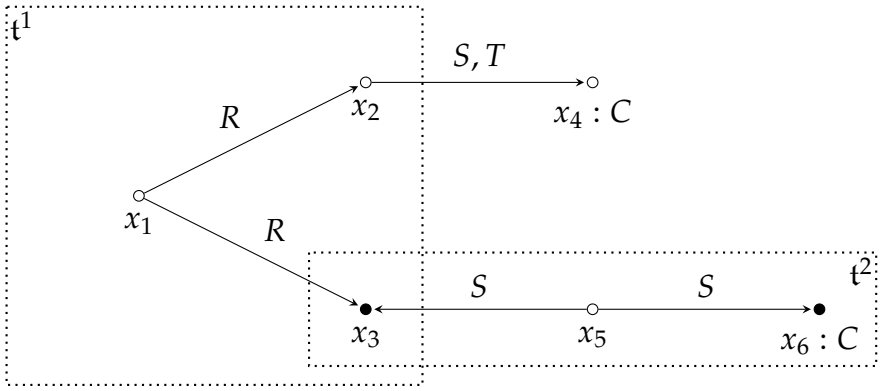

Figure 5.2: Multigraph representation of the conjunctive query $\{C(x_4), C(x_6), R(x_1, x_2), R(x_1, x_3), S(x_2, x_4), S(x_5, x_3), S(x_5, x_6), T(x_2, x_4)\}$, with distinguished variables x_3 and x_6 marked by solid, black dots. Two tree-witnesses are marked in the figure: $\mathfrak{t}^1 = \langle\{x_2, x_3\}, \{x_1\}\rangle$ and $\mathfrak{t}^2 = \langle\{x_3, x_6\}, \{x_5\}\rangle$. The corresponding tree-witness queries are $\mathbf{tw}_{\mathfrak{t}^1} = \exists z[A_{R^-}(z) \wedge (x_2 = z) \wedge (x_3 = z)]$ and $\mathbf{tw}_{\mathfrak{t}^2} = \exists y[A_{S^-}(y) \wedge (x_3 = y) \wedge (x_6 = y)]$.

Figure 5.2 shows how the query can be divided into consistent tree-witnesses. These tree-witnesses are exactly the parts that must be mapped to some witness tree in the canonical model multigraph. Using the tree-witnesses in the figure we end up with the query

$$\exists z\big[A_{R^-}(z) \wedge A_{S^-}(z) \wedge S(z, x_4) \wedge T(z, x_4) \\ \wedge C(z) \wedge C(x_4) \wedge (x_2 = z) \wedge (x_3 = z) \wedge (z_6 = z)\big],$$

where x_1 and x_5 have been removed altogether. □

In Example 5.4.1, we matched $R(x_1, x_2) \wedge R(x_1, x_3)$ to $\mathbf{q}_{\mathfrak{t}^1}$, and $S(x_5, x_3) \wedge S(x_5, x_6)$ to $\mathbf{q}_{\mathfrak{t}^2}$ (where $\mathbf{q}_{\mathfrak{t}^1}$ and $\mathbf{q}_{\mathfrak{t}^2}$ are the tree-witnesses of Figure 5.2). We then made sure that the roots of $\mathbf{q}_{\mathfrak{t}^1}$ and $\mathbf{q}_{\mathfrak{t}^2}$ are the same, that they are in C, and that they are both S- and T-related to some individual in C.

The elegance of this approach is its compatibility with the virtual ABox. When matching a query to some $\mathcal{C}_\mathcal{K}$, we need to look for ways to match our tree-witnesses to restrictions $\mathcal{C}^R_\mathcal{K}(a)$. The answer variables of the subquery that we match to $\mathcal{C}^R_\mathcal{K}(a)$, need to be matched to an ABox individual that are guaranteed to be R-related to some individual. In the virtual ABox, these individuals are exactly the individuals of A_R (the domain of R).

Since our ABox is H-complete, we do not need to make an ontology rewriting of $A_R(z)$. All the relevant inclusion axioms have been taken care of through H-completion (see Definitions 3.5.3 and 4.3.1). The inclusion axioms that are not taken care of by H-completeness (some concept is included in the domain of some relation), are exactly the axioms taken care of by tree-witnesses.

5.4.1 Brute force tree-witness search

The simplest tree-witness search is a brute force search. In such a search, we construct every possible pair $\langle t_r, t_w \rangle$ that could be a tree-witness, and check if it satisfies Definition 5.3.2.

Given a conjunctive query **cq**:

1. for every role atom $P(x, y)$ in **cq** with $x \in \mathrm{dvar}(\mathbf{cq})$, construct every possible pair $t = \langle t_r, t_w \rangle$ satisfying point 1 of Definition 5.3.2, starting with the smallest sets t_r and t_w. Then, for each pair,
   a) apply point 2 of Definition 5.3.2 to fixpoint, discard t if point 1 is no longer satisfied, otherwise,
   b) if we have previously found a tree-witness t' such that $t'_r \subseteq t_r$ and $t'_w \subseteq t_w$, discard t, otherwise
   c) check if $\mathcal{C}^P_{\mathcal{K}}(a) \models \mathbf{q}_t\sigma$ for some σ (point 3 of Definition 5.3.2), if not, discard t.

The tree-witnesses for **cq** are the pairs $\langle t_r, t_w \rangle$ that are not discarded.

The search for tree-witnesses depends on the knowledge base $\mathcal{K}$, but it does not require that we generate the (possibly infinite) canonical model $\mathcal{C}_{\mathcal{K}}$. The generating relation $\rightsquigarrow$ from Algorithm 5.1.2 is sufficient, and it can be computed in polynomial time in the size of $\mathcal{K}$ [KKZ11].

5.5 Complexity of the tree-witness rewriting

The tree-witness rewriting is intractable. Since the number of combinations of tree-witnesses may be exponential in the size of the input query, the tree-witness rewriting is not even in NP.

We have shown, in Section 5.2.3, that finding a homomorphism from a query to the canonical model with materialised ABoxes is NP-hard. In the tree-witness rewriting, we do not need to find a homomorphism

from the entire query into the canonical model. Instead, we need to find homomorphisms from parts of the query into the restrictions of the canonical model, and then combine these into an ontology rewriting.

Since the tree-witness rewriting is an ontology rewriting for OWL 2 QL ontologies, it is generally intractable [KKZ11]. However, in [KKZ11], Kikot et al. shows that checking if a tree-shaped query matches the canonical model can be done in time polynomial in $|\mathcal{T}|$. A key result leading up to the polynomial matching, is that the generating relation $\rightsquigarrow$ from Definition 5.1.1 can be computed in time polynomial in $|\mathcal{T}|$.

The intractability of the tree-witness rewriting turns out to be caused by the fact that the number of homomorphisms that must be checked (in our case the tree-witnesses) is bounded by a function exponential in $|\mathbf{q}|$. The size of the tree-witness rewriting may also be exponential in $|\mathbf{q}|$, as the following example from [KKZ12] shows.

Example 5.5.1. Let $\mathbf{q}^N(x)$ be the conjunctive query

$$\mathbf{q}^N(x) = \{R_i(x, y_i) \mid i \leq N\},$$

where R_i are distinct roles, and x is the only answer variable. There are N tree-witnesses $\langle\{x\}, \{y_i\}\rangle$ for $\mathbf{q}^N$, all of which are pairwise consistent. The tree-witness rewriting is

$$\mathbf{q}^N_{\mathbf{tw}}(x) = \bigvee_{J \subseteq [0,N]} \left(\left[\bigwedge_{i \in J} A_{R_i}(x) \right] \wedge \left[\bigwedge_{i \notin J} R_i(x, y_i) \right] \right).$$

The size of $\mathbf{q}^N_{\mathbf{tw}}$ is exponential in the size of $\mathbf{q}^N$, because for every role in $\mathbf{q}^N$, we have two options: answer by tree-witness or answer by ABox.

An alternative linear size rewriting is

$$\mathbf{p} = \bigwedge_{i \leq N} \left(A_{R_i} \vee R(x, y_i) \right),$$

but this rewriting simply moves the exponential cost of the joins to the database query execution stage. This takes away our control over further optimization, and places more load on a central piece of the architecture. For these two reasons, the linear rewriting $\mathbf{p}$ is usually not an acceptable solution. □

5.5.1 Queries with few tree-witnesses

The tree-witness rewriting is generally intractable, but it turns out to be very effective for many natural query structures [RKZ13]. We wish to identify conditions that limit the number of possible tree-witnesses, as the potentially exponential number of tree-witnesses is the weakest link in the tree-witness rewriting.

Disconnected queries

If the multigraph representation of the query is not connected, except by edges between answer variables, then we can treat the connected subgraphs separately in the tree-witness search. The total time required to search for tree-witnesses is then the sum of the times required for the smaller queries. If the query $\mathbf{q}$ can be split into k equal parts, we get a running time bound by $O(c^{|\mathbf{q}|/k})$ instead of $O(c^{|\mathbf{q}|})$.

Example 5.5.2. Consider the query

$$\mathbf{q}(x_1, x_2) = R(x_1, x_2) \wedge S(x_1, x_3) \wedge T(x_2, x_4).$$

Since atoms in tree-witnesses must contain at least one non-distinguished variable (see the construction of $\mathbf{q}_t$ in Definition 5.3.2), x_3 and x_4 cannot be in the same tree-witness. □

Non-distinguished variables

When finding tree-witnesses, we must always choose distinguished variables to be roots of the restricted canonical models. The non-distinguished variables, on the other hand, can be mapped to both roots and witness parts in the canonical model. As such, limiting the amount of non-distinguished variables will reduce the tree-witness search time more efficiently than limiting the amount of distinguished variables.

Example 5.5.3. We revisit the query of Example 5.5.1:

$$\mathbf{q}^N(x) = \{R_i(x, y_i) \mid i \leq N\}.$$

We now alter this query by replacing y_j with a distinguished variable z_j for $j \leq M < N$. We let $\vec{z} = \langle z_1, \ldots, z_M \rangle$, and get the query

$$\mathbf{q}^N(x, \vec{z}) = \{R_i(x, z_i) \mid i \leq M\} \cup \{R_i(x, y_i) \mid M < i \leq N\}.$$

The tree-witness rewriting becomes

$$\mathbf{q}^{N}_{\mathbf{tw}}(x,\vec{z}) = \bigvee_{J\subseteq[M+1,N]} \left(\left[\bigwedge_{i\in J} A_{R_i}(x) \right] \wedge \left[\bigwedge_{i\notin J} R_i(x,y_i) \right] \wedge \left[\bigwedge_{i\leq M} R_i(x,z_i) \right] \right).$$

The new query is significantly smaller than the old, since the number of disjuncts is exponential in $N - M$. □

In this chapter, we have described the tree-witness rewriting over H-complete ABoxes. In particular, we have studied what parts of this rewriting are particularly resource intensive. In the next chapter, we try to use the perfect mappings from Chapter 4 to reduce the cost of the tree-witness rewriting.

Chapter 6

Perfect mapping tree-witness rewriting

In this chapter we present the bulk of our independent results. We show how to combine the application of perfect mappings (Section 4.6) with the tree-witness rewriting (Chapter 5).

6.1 Tree-witness rewriting with perfect mappings

Recall (Definitions 5.3.2 and 5.3.5) that a tree-witness for a query $\mathbf{q}$ is a pair $\langle t_r, t_w \rangle$, with a corresponding tree-witness query $\mathbf{q}_t$, such that $\mathbf{q}_t$ contains every atom of $\mathbf{q}$ with non-distinguished variables in t_w.

A tree-witness specifies a part of the query that can be replaced by a much simpler query. The replacement is sound, because the TBox guarantees the fulfilment of some part of the query, on the condition that the rest of the query is satisfied. The part that is guaranteed for by the TBox, can then be removed. Tree-witnesses are not, however, complete. They only provide answers corresponding to one specific substructure in the canonical model of the knowledge base. Multiple tree-witnesses must be used in order to find all the answers to a query.

Using perfect mappings, we can make much stronger (and in fact complete) tree-witnesses.

Definition 6.1.1 (Perfect mapping tree-witness)**.** Let $\mathcal{B} = \langle \mathcal{O}, \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$ be an OBDA specification, $\mathcal{M}_P$ a perfect mapping for $\mathcal{B}$, and $\mathbf{q}$ a query over $\mathcal{B}$. Let $\mathbf{p}$ be a subset of $\mathbf{q}$ such that $\mathbf{p}$ and $\mathbf{q} \setminus \mathbf{p}$ do not share non-

distinguished variables. If $m : Q \rightsquigarrow \mathbf{p}'$ is a mapping assertion in $\mathcal{M}_P$, and σ is a homomorphism (Definition 5.2.2) from $\mathbf{p}'$ to $\mathbf{p}$, such that $\mathbf{p} = \mathbf{p}'\sigma$, then

$$\mathfrak{t}^m = \langle \mathrm{dvar}(\mathbf{p}), \mathrm{nvar}(\mathbf{p}) \rangle, \qquad \mathbf{q}_{\mathfrak{t}^m} = \mathbf{p}$$

is the *perfect mapping tree-witness generated by m and* $\mathbf{p}$.

Two perfect mapping tree-witnesses $\mathfrak{t}^{m_1}$ and $\mathfrak{t}^{m_2}$ are consistent if $\mathbf{q}_{\mathfrak{t}^{m_1}}$ and $\mathbf{q}_{\mathfrak{t}^{m_2}}$ are disjoint. A perfect mapping tree-witness $\mathfrak{t}^m$ and a regular tree-witness $\mathfrak{t}$ are consistent if $\mathfrak{t}^m_w$ and $\mathfrak{t}_w$ are disjoint. □

Perfect mapping tree-witnesses are different from regular tree-witnesses in that they do not require all the answer variables in $\mathfrak{t}^m_r$ to be assigned the same individual. The tree-witness query for $\mathfrak{t}^m$ is an n-ary relation preserving all the constraints expressed in $\mathbf{p}$, namely the body of m. Where a regular tree-witness must match some $\mathcal{C}^R_{\mathcal{K}}(a)$, a perfect mapping tree-witness does not need to match the canonical model at all. A perfect mapping is guaranteed to provide ABox individuals for each of its distinguished variables, and witnesses for its non-distinguished variables, without referring to the TBox.

A more liberal application of perfect mappings could, for example, drop the requirement that $\mathbf{p}$ and $\mathbf{q} \setminus \mathbf{p}$ do not share non-distinguished variables. The problem is that such a use of perfect mappings could lead to violations of Theorem 6.1.3. If Theorem 6.1.3 does not hold, then Theorem 6.1.4 does not hold, and we can create a counter example to the key result of this section: Theorem 6.1.7 (the perfect mapping tree-witness rewriting is a complete ontology rewriting over H-complete ABoxes). Example 6.1.8 is such a counter example.

Definition 6.1.2 (Perfect mapping tree-witness query)**.** Let $\mathfrak{t}^m$ be the perfect mapping tree-witness generated by the perfect mapping assumption $m : Q(\vec{x}, \vec{y}) \rightsquigarrow q(\vec{x})$ and some query $\mathbf{p}$. The *perfect mapping tree-witness query generated by m and* $\mathbf{p}$ is $\mathbf{tw}_{\mathfrak{t}^m} = v(\vec{x})$, where v is the view predicate in m^L and m^H.

The perfect mapping tree-witness query does not need to contain the equality constraints of regular tree-witness queries. This is because all the original names of the distinguished variables are kept.

The next theorem shows that if a regular tree-witness $\mathfrak{t}$ intersects a perfect mapping tree-witness $\mathfrak{t}^m$, then the query $\mathbf{q}_{\mathfrak{t}}$ is contained in $\mathbf{q}_{\mathfrak{t}^m}$.

Theorem 6.1.3. *Let* $\mathfrak{t}$ *be a tree-witness for* $\mathbf{q}$*, and* $\mathfrak{t}^m$ *be a perfect mapping tree-witness for* $\mathbf{q}$ *generated by some perfect mapping assertion* m *and some subquery* $\mathbf{p}$ *of* $\mathbf{q}$*. If* $\mathfrak{t}$ *and* $\mathfrak{t}^m$ *are not consistent, then* $\mathbf{q}_\mathfrak{t}$ *is a subquery of* $\mathbf{q}_{\mathfrak{t}^m}$*.*

Proof. Since $\mathbf{q}_\mathfrak{t}$ and $\mathbf{q}_{\mathfrak{t}^m}$ share at least one non-distinguished variable, the containment follows from the $\subseteq$-minimality of $\mathbf{q}_\mathfrak{t}$, and the construction of $\mathbf{q}_\mathfrak{t}$ and $\mathbf{q}_{\mathfrak{t}^m}$. Point 2 in Definition 5.3.2 defines what parts of the full query must be part of a tree-witness. The same requirement holds for perfect mapping tree-witnesses, but perfect mapping tree-witnesses have no minimality constraint. Therefore, each atom in $\mathbf{q}_\mathfrak{t}$ must also be an atom of $\mathbf{q}_{\mathfrak{t}^m}$ by their constructions, but not necessarily vice versa. □

Theorem 6.1.3 is the reason why the perfect mapping tree-witness rewriting is complete. As long as $\mathbf{q}_\mathfrak{t}$ is a subquery of $\mathbf{q}_{\mathfrak{t}^m}$, we know that each answer to $\mathbf{q}_\mathfrak{t}$ must also be an answer to $\mathbf{q}_{\mathfrak{t}^m}$ (by definition of a perfect mapping assertion). However, if $\mathbf{q}_\mathfrak{t}$ only intersected $\mathbf{q}_{\mathfrak{t}^m}$, then $\mathbf{q}_\mathfrak{t}$ could provide answers to parts of the query that $\mathbf{q}_{\mathfrak{t}^m}$ does not cover. As such, we could not discard $\mathbf{q}_\mathfrak{t}$.

Using Theorem 6.1.3, we can now prove the completeness of the perfect mapping tree-witness rewriting.

Theorem 6.1.4. *Let* $\mathcal{B} = \langle \mathcal{O}, \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$ *be an OBDA specification with a perfect mapping* $\mathcal{M}_P$*,* $\mathbf{q}$ *a conjunctive query over* $\mathcal{B}$*,* Θ_P *a consistent set of perfect mapping tree-witnesses for* $\mathbf{q}$*, and* Θ *a consistent set of regular tree-witnesses for* $\mathbf{q}$*. Let*

$$\Theta' = \Theta_P \cup \{\mathfrak{t} \in \Theta \mid \Theta_P \cup \{\mathfrak{t}\} \textit{ is consistent}\},$$

and define, for any consistent set of tree-witnesses X*,*

$$\mathbf{q}^X = (\mathbf{q} \setminus \mathbf{q}_X) \wedge \bigwedge_{\mathfrak{t} \in X} \mathbf{tw}_\mathfrak{t},$$

where $\mathbf{q}_X = \{\mathbf{q}_\mathfrak{t} \mid \mathfrak{t} \in X\}$*, then* $\mathit{CertAns}(\mathbf{q}^\Theta, \mathcal{B}, D) \subseteq \mathit{CertAns}(\mathbf{q}^{\Theta'}, \mathcal{B}, D)$ *for any instance* D *of* $\mathcal{S}$*, so long as* $\mathcal{M}_P^L \subseteq \mathcal{M}^\mathcal{T}$*.*

Proof. Note that $\mathbf{q}_\Theta$ is a subquery of $\mathbf{q}_{\Theta'}$, since by Theorem 6.1.3, any $\mathfrak{t} \in \Theta$ that comes into conflict with one of the $\mathfrak{t}^m \in \Theta_P$ is such that $\mathbf{q}_\mathfrak{t}$ is a subquery of $\mathbf{q}_{\mathfrak{t}^m}$.

Let m be some perfect mapping assertion, such that $\mathfrak{t}^m \in \Theta_P$. The tree-witness query $\mathbf{tw}_{\mathfrak{t}^m}$ replaces, in $\mathbf{q}^\Theta$, zero or more queries $\mathbf{tw}_\mathfrak{t}$, and

possibly some atoms in $\mathbf{q} \setminus \mathbf{q}_\Theta$. The query $\mathbf{p}$, the head of m, is the union of tree-witness queries and other atoms replaced by $\mathbf{tw}_{t^m}$. Since m is a perfect mapping assertion, any rewriting of the $\mathbf{p}$ must be contained in $\mathbf{tw}_{t^m}$ (by the definition of perfect mappings, Definition 4.6.2). Thus, the answers to any query removed when replacing Θ with Θ', are also answers to the perfect rewriting replacing them. □

Since we do not need to use tree-witnesses that intersect a perfect mapping tree-witness, we extend the definition of consistent sets of tree-witnesses.

Definition 6.1.5 (Restricted, consistent sets of tree-witnesses)**.** Let $\mathcal{B} = \langle \mathcal{O}, \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$ be an OBDA specification, $\mathbf{q}$ a conjunctive query over $\mathcal{B}$, Θ_P a consistent set of perfect mapping tree-witnesses for $\mathbf{q}$, and Θ' as in Theorem 6.1.4. We then define

$$\Xi = \{\Theta' \mid \Theta \text{ is a consistent set of tree-witnesses for } \mathbf{q}\},$$

the *consistent sets of tree-witnesses restricted to* Θ_P. □

Note that Ξ may be significantly smaller than the set of all consistent sets of tree-witnesses. This is because the perfect mapping tree-witnesses may eliminate many different combinations of regular tree-witnesses.

Ξ will not be unique if there are non-consistent perfect mapping tree-witnesses. In such a case we use some selection rule to decide which perfect mapping tree-witnesses to include in Θ_P.

Definition 6.1.6 (Perfect mapping tree-witness rewriting)**.** Let $\mathcal{B} = \langle \mathcal{O}, \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$ be an OBDA specification with a perfect mapping $\mathcal{M}_P$, $\mathbf{q}$ be a conjunctive query over $\mathcal{B}$, Θ_P a consistent set of perfect mapping tree-witnesses for $\mathbf{q}$, and Ξ the consistent sets of tree-witnesses restricted to Θ_P. Then

$$\mathbf{q}^P_{\mathbf{tw}} = \bigvee_{\Theta \in \Xi} \exists \vec{y} \left[(\mathbf{q} \setminus \mathbf{q}_\Theta) \wedge \bigwedge_{t \in \Theta} \mathbf{tw}_t \right]$$

is a *perfect mapping tree-witness rewriting of* $\mathbf{q}$, $\mathcal{T}$, *and* $\mathcal{M}_P$. □

Theorem 6.1.7. *Let* $\mathcal{B} = \langle \mathcal{O}, \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$ *be an OBDA specification with a perfect mapping* $\mathcal{M}_P$*, and* $\mathbf{q}$ *be a conjunctive query over* $\mathcal{B}$*. Then* $\mathbf{q}^P_{\mathbf{tw}}$ *is an ontology rewriting of* $\mathbf{q}$ *and* $\mathcal{T}$ *for H-complete virtual ABoxes, as long as* $\mathcal{M}^L_P \subseteq \mathcal{M}^{\mathcal{T}}$.

Proof. From Theorem 5.3.8, we know that the tree-witness rewriting is an ontology rewriting over H-complete ABoxes. From Theorem 6.1.4, we know that we can safely replace the consistent sets of tree-witnesses with the consistent sets of tree-witnesses restricted to sets including some perfect mapping tree-witnesses. □

As discussed after Definition 6.1.1, the perfect mapping tree-witness rewriting only works if we are careful about how we apply our perfect mapping tree-witnesses.

Example 6.1.8. We look at the perfect mapping tree-witness rewriting of the query

$$\mathbf{q}(x_1) = \{A(x_1), R(x_1, x_2), S(x_2, x_3)\},$$

where we drop the requirement that $\mathbf{p}$ (a subquery of $\mathbf{q}$) and $\mathbf{q} \setminus \mathbf{p}$ do not share non-distinguished variables (from Definition 6.1.1). We assume the TBox axioms

$$A \sqsubseteq \exists R, \qquad \exists R^- \sqsubseteq \exists S,$$

and the perfect mapping assertion

$$m \quad : \quad Q(x) \quad \rightsquigarrow \quad A(x) \wedge R(x, y)$$

with

$$\begin{array}{lcll} m^H & : & v(x) & \rightsquigarrow \quad A(x) \wedge R(x, y) \\ m^L & : & Q(x) & \rightsquigarrow \quad v(x), \end{array}$$

and regular mapping assertions

$$\begin{array}{lcll} m_1 & : & Q_A(x) & \rightsquigarrow \quad A(x) \\ m_2 & : & Q_R(x, y) & \rightsquigarrow \quad R(x, y) \\ m_3 & : & Q_S(x, y) & \rightsquigarrow \quad S(x, y). \end{array}$$

A perfect mapping tree-witness rewriting would now encounter a problem with the variable x_2. The perfect mapping tree-witness guarantees for the x_2 in $R(x_1, x_2)$, but does not provide an answer for it, so we cannot query the ABox for an answer to $S(x_2, x_3)$. The remaining way to find answers to $S(x_2, x_3)$ is to use the tree-witness $\langle \{x_1\}, \{x_2, x_3\} \rangle$,

but this tree-witness is not consistent with our perfect mapping tree-witness. The perfect mapping tree-witness rewriting stops at

$$Q(x) \wedge S(x_2, x_3),$$

where we have no legal way of rewriting $S(x_2, x_3)$, since there is an unknown constraint on x_2.

If we reinstate all the requirements of Definition 6.1.1, the only perfect mapping tree-witness we have is not applicable, and we get the (perfect mapping) tree-witness rewriting

$$\big[Q_A(x_1)\big] \vee \exists x_2, x_3 \big[Q_A(x_1) \wedge Q_R(x_1, x_2) \wedge Q_S(x_2, x_3)\big],$$

where the first disjunct is a result of using the only tree-witness, and the second is the result of using no tree-witnesses. □

6.2 Algorithm outline

We now combine all our results into an algorithm outline. In each step, we refer to the definitions relevant to that step.

Input: An OBDA specification $\mathcal{B} = \langle \mathcal{O}, \mathcal{T}, \mathcal{S}, \mathcal{M} \rangle$, a perfect mapping $\mathcal{M}_P$ from $\mathcal{S}$ to $\mathcal{O}$, and a conjunctive query $\mathbf{q}$ over $\mathcal{B}$.

Output: A union of conjunctive queries $\mathbf{cq}$ that is an ontology rewriting of $\mathbf{q}$ and $\mathcal{T}$ for H-complete ABoxes.

Step 1: Find perfect mapping tree-witnesses

Use $\mathcal{M}_P$, $\mathbf{q}$ and some heuristic to create a maximal consistent set Θ_P of perfect mapping tree-witnesses (Definition 6.1.1).

Step 2: Find the consistent sets of tree-witnesses restricted to Θ_P

Use Θ_P to find Ξ, the set of all the consistent sets of tree-witnesses that contain at least all the perfect mapping tree-witnesses in Θ_P (Definitions 5.3.2 and 6.1.5).

Step 3: Compute the ontology rewriting

Use Ξ to compute the ontology rewriting $\mathbf{cq} = \mathbf{q}^P_{\mathbf{tw}}$ (Definitions 5.3.7, 6.1.2 and 6.1.6).

□

6.3 Simplifying the $\mathcal{T}$-mapping with perfect mappings

Given a perfect mapping $\mathcal{M}_P$ and a $\mathcal{T}$-mapping $\mathcal{M}^{\mathcal{T}}$, we can create a new $\mathcal{T}$-mapping $\mathcal{M}_P \cup \mathcal{M}^{\mathcal{T}}$, and simplify using Theorem 4.4.3 (simplification by query containment), and the fact that a perfect mapping assertion makes redundant all other regular mapping assertions whose heads are the same (down to a substitution on the head of the perfect mapping).

This technique only works with perfect mapping assertions with atomic heads, since this is when the other mapping assertions may have matching heads.

On the other hand, the perfect mapping assertions with non-atomic heads will have a larger impact on the tree-witness rewriting. The larger queries can invalidate more tree-witnesses, and thus make the rewriting smaller. Perfect mapping assertions with non-atomic heads can also be used in the unfolding stage of each individual query [PLL+13].

When creating a perfect mapping tree-witness rewriting, we must choose a consistent set of perfect mapping tree-witnesses. Finding an optimal such set is very hard. In fact, the number of combinations of perfect mapping tree-witnesses may be exponential in the number of perfect mapping assertions (by the same argument as for regular tree-witnesses, see Example 5.5.1).

A greedy strategy, suggested by [PLL+13], is to order all the perfect mapping assertions by their lengths. We try the longest first.

The length of a mapping assertion length(m), is suggested by [PLL+13] to be the number of atoms in the head of m. Since we want to eliminate possible tree-witnesses, we may want to instead count the number of role atoms in the head of m. In this latter case, we could order first by the number of role-atoms, and then by the total number of atoms. We further discuss the choice of perfect mapping assertions in Section 6.4.2.

6.4 Complexity assessment

The perfect mapping tree-witness rewriting consists of two major parts: preparation of the mapping, and the rewriting procedure itself.

The preparation of the mapping is a simple procedure, especially the

steps required to make it ready for perfect mapping tree-witnesses (adding $\mathcal{M}_P^L$). The time consuming steps of this part is the simplification of the $\mathcal{T}$-mapping, discussed in Section 4.4. Since the preparation of the mapping only needs to be done when the mapping or knowledge base changes, we do not study this part in more detail.

The perfect mapping tree-witness rewriting is further divided into two stages: finding perfect mapping tree-witnesses, and finding ordinary tree-witnesses.

As discussed in Section 5.4, finding the tree-witness rewriting of a query is intractable. The main problem is that the number of possible tree-witnesses that must be inspected is only exponentially bounded in the size of the query.

Given a consistent set of perfect mapping tree-witnesses for a query, there is a large potential for reducing the time and memory required by the ordinary tree-witness rewriting, and also for shortening the final rewriting.

As we state in Theorem 6.1.3, no tree-witness may intersect a perfect mappping tree-witness. That means every atom contained in some query $\mathbf{q}_{\mathfrak{t}^P}$, where $\mathfrak{t}^P$ is a perfect mapping tree-witness, may be removed from the original query before we search for tree-witnesses. A reduction in the size of $|\mathbf{q}|$ may dramatically reduce the time required to search for tree-witnesses.

Example 6.4.1. We take another look at the query in Example 5.5.1, but we now assume we have a set of perfect mapping assertions

$$\{m_i : Q_i(z_1, z_2) \rightsquigarrow R_i(z_1, z_2) \mid i \leq k\}.$$

We then get the perfect mapping tree-witness rewriting

$$(\mathbf{q}_{\mathbf{tw}}^N)^P(x) = \bigvee_{J \subseteq [k+1,N]} \left(\left[\bigwedge_{i \in J} A_{R_i}(x) \right] \wedge \left[\bigwedge_{i \notin J} R_i(x, y_i) \right] \wedge \left[\bigwedge_{i \leq k} Q_i(x, y_i) \right] \right).$$

The perfect mapping rewriting is still a DNF formula with clause length N, but the number of clauses is 2^{N-k}, instead of 2^{N+1}. So the rewriting has been shortened by a factor of 2^{k+1}. □

6.4.1 Finding perfect mapping tree-witnesses

Finding perfect mapping tree-witnesses for a query $\mathbf{q}$, amounts to finding homomorphisms from the heads of perfect mapping assertions to $\mathbf{q}$. This process is generally intractable (NP-complete), but the input to this step is much smaller than the input to the search for ordinary tree-witnesses (a problem which is harder than NP, due to the potentially exponential size of the rewriting, see Example 5.5.1).

Given an algorithm that is good at finding homomorphisms, we can spend time running this algorithm, and in exchange, reduce the amount of time spent later on the search for ordinary tree-witnesses.

6.4.2 Choosing perfect mapping tree-witnesses

We took a brief look at heuristics for choosing perfect mapping assertions in Section 6.3. We now revisit that topic, and discuss how to combine the observations of Sections 5.5.1 and 6.4.1.

Instead of just the number of role atoms and the total number of atoms, we now want to consider a wider range of parameters for our heuristics.

Number of role atoms: denoted N_R, the number of role atoms in the head of the assertion.

Number of concept atoms: denoted N_C, the number of concept atoms in the head of the assertion.

Splitting: denoted N_S, the (approximate) size of the smallest subquery if splitting occurs, 0 otherwise.

Non-distinguished variables: denoted N_V, the number of different non-distinguished variables in the head of the assertion.

We suggest weighted sums of these parameters as possible heuristics. N_R should be heavily weighted, and may even be the only parameter used for initial sorting, with the other parameters used for tie-breaking.

We discuss each parameter separately.

Number of role atoms, concept atoms, and non-distinguished variables

These measures are all counts of syntactic elements in the query. As such, they can all be computed in time linear in the query size.

The number of role atoms deserves special attention; it is very indicative of quality, and at the same time very easy to calculate.

Splitting into disconnected queries

The measure of splitting must be estimated; an exact calculation of this measure requires actually finding a homomorphism.

We wish to check if our query graph can be split into two graphs by the removal of a single edge (role atom). If this is the case, we wish to base N_S on how much this splitting gains us, and how likely it is that a matching edge in the head of a perfect mapping assertion will be mapped to the splitting edge, and not a different edge with the same name.

Consider a graph G of size M, consisting of two connected subgraphs G_1 and G_2 of size M_1 and M_2, such that $M_1 < M_2$. Let the subgraphs be connected by a single edge $P(x, y)$. Suppose the head $\mathbf{p}$ of a perfect mapping assertion contains an atom P. One possible way to define N_S is then

$$N_S = c(M_1 - |\mathbf{p}|),$$

where c is some weight, for example one divided by the number of P-atoms in the entire query.

If c equals 1, then the above definition makes N_S the minimal reduction in query size of the queries to be manipulated further. Originally, we had to find tree-witnesses for the graph G. After applying our perfect mapping, the size of G is reduced by $|\mathbf{p}|$.

Because of the splitting, the largest of the two remaining graphs is no larger than $M - M_1$. Without splitting, the size reduction would still be $|\mathbf{p}|$, so the size reduction (difference in size between the largest subquery in each case), is $M_1 - |\mathbf{p}|$.

The weight c should be less than 1 to reflect the possibility that a match between $\mathbf{p}$ and G does not need to contain the P atom connecting the subgraphs, for example 1 divided by the number of P atoms in the query. For perfect mapping assertions with large heads, c could be the number of P atoms in $\mathbf{p}$ over the number of P atoms in the query.

Checking if a graph G is connected can be done in time $O(|G|^2)$. Removing every edge, one at a time, to find edges that can disconnect the graph (and checking the sizes of the resulting subgraphs), can be done in $O(|G|^3)$. Thus, estimating N_S in the way we suggest is $O(|G|^3)$.

6.4.3 Complexity trade-off

Example 6.4.1 shows that there is great potential gain from using perfect mappings in the tree-witness rewriting.

Although both ontology rewriting and matching perfect mapping assertions to queries are intractable problems, the latter is in NP (finding homomorphisms), while the former is not. Besides, [PLL+13] have provided examples where perfect mappings provide drastic improvements.

A key concern with the perfect mapping tree-witness rewriting is the cost of using it when no perfect mapping assertions can be applied. The effectiveness of the perfect mapping tree-witness rewriting relies heavily on a good heuristic for finding matching perfect mapping assertions.

Since it is not necessary for correctness to find all matching perfect mapping assertions, we may not want to use an algorithm that is guaranteed to find every match. Instead, we could use an algorithm that terminates quickly if no match is likely.

Chapter 7

Summary

7.1 Conclusions

Using perfect mappings [PLL+13], we have made an improvement to the tree-witness rewriting over H-complete virtual ABoxes [RKZ13]. Our analysis indicates that our algorithm, outlined in Section 6.2, can perform well given frequent matches between the input queries and the perfect mapping assertions.

The efficiency of our perfect mapping tree-witness rewriting relies heavily on an efficient homomorphism finder, preferably one that can be set up to terminate quickly if finding a homomorphism is unlikely.

As discussed in Section 6.4, the perfect mapping tree-witness rewriting may outperform the regular tree-witness rewriting significantly if large portions of the query is matched to perfect mappings. The drawback is that the perfect mapping tree-witness rewriting must spend time looking for perfect tree-witnesses also when it ends up defaulting to the regular rewriting.

The perfect mapping tree-witness rewriting is most useful on large ontology queries over OBDA specifications with well curated perfect mappings. The perfect mappings should not contain assertions that are rarely matched, or assertions where it is very difficult to determine a match.

Further studies are required before we can say if our results are of interest to the Optique project (see [KJZ+13]).

7.2 Future work

In Sections 5.4 and 6.4 we discuss some conditions under which the perfect mapping tree-witness rewriting is very advantageous. Aside from further exploration of these conditions, we single out three natural steps forward: simulations with our proposed methods, studies of optimal conditions for the unfolding step, and studies of perfect mappings themselves.

7.2.1 Simulation

On their own, the tree-witness rewriting and the perfect mapping approach have performed well in simulations on a broad selection of real world ontologies (see [RKZ13] and [PLL+13] respectively).

In the perfect mapping tree-witness rewriting (Chapter 6) we make one-time improvements on the mapping (achieving H-completeness), improving query answering performance for subsequent queries at a cost constant in the number of queries rewritten. The introduction of perfect mappings into the tree-witness rewriting process potentially invalidates many tree-witnesses, but at the cost of adding a computational step in order to find suitable perfect tree-witnesses. This added cost must be paid for every query that is answered.

We propose simulation as a natural step forward. With simulations, we will not only be able to measure the performance of the perfect tree-witness rewriting, but also the effectiveness of different heuristics for choosing the set of perfect tree-witnesses.

7.2.2 Unfolding

Both Pinto et al. [PLL+13] and Rodríguez-Muro et al. [RKZ13] suggest optimizations of the unfolding step. [PLL+13] suggest pre-analysing the bodies of the mappings to find containments or special cases where we have containment. [RKZ13] suggest checking for inclusion dependencies (foreign keys) in each step of the unfolding. These dependencies can be quite common due to reification of n-ary relations (which are themselves common due to Description logics limitation to binary roles).

In this work, we have paid very little attention to the unfolding process. In particular, to how our choices in rewriting affect this process.

A possible future line of inquiry is looking for patterns that can be used

to improve the heuristics of Section 6.4, with respect to the entire query rewriting (ontology rewriting and unfolding combined).

7.2.3 Finding perfect mappings

Perfect mappings have two major sources: expert knowledge of the OBDA scheme, and cached unfoldings. Both of these provide perfect mapping assertions that require maintenance.

As our first source, an expert on an OBDA system can write mapping assertions that are known to be perfect given the domain of interest, the ontology and the database schema. These mapping assertions must undergo maintenance by an expert whenever the OBDA specification changes.

Our second source of perfect mappings are cached unfoldings. Whenever a query is unfolded, we know that there is a perfect mapping assertion from the unfolded query to the original query, as long as the base for the unfolding remains the same. These mapping assertions must also undergo maintenance whenever the ontology or the mapping changes (see for example [GMS93]). However, this maintenance may be automated, since these mapping assertions are based only on the OBDA specification, and not on external knowledge.

In [PLL+13], Pinto et al. use the second approach, resulting in an unfolding algorithm that improves with use.

We suggest studying how to maintain a set of cached unfoldings, so that they can be used as perfect mappings. Since perfect mapping assertions may be costly to maintain, and the size of the number of perfect mapping assertions affects the running time of the perfect mapping tree-witness rewriting, we may need to curate the perfect mapping carefully.

Bibliography

[CDL+06] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. In P. Doherty, J. Mylopoulos, and C. A. Welty, editors, *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006*, pages 260–270. AAAI Press, 2006.

[CDL+07] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. Autom. Reasoning*, 39(3):385–429, 2007.

[GMS93] A. Gupta, I. S. Mumick, and V. S. Subrahmanian. Maintaining views incrementally. In P. Buneman and S. Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, 1993.*, pages 157–166. ACM Press, 1993.

[KGJ+13] E. Kharlamov, M. Giese, E. Jiménez-Ruiz, M. G. Skjæveland, A. Soylu, D. Zheleznyakov, T. Bagosi, M. Console, P. Haase, I. Horrocks, S. Marciuska, C. Pinkel, M. Rodriguez-Muro, M. Ruzzi, V. Santarelli, D. F. Savo, K. Sengupta, M. Schmidt, E. Thorstensen, J. Trame, and A. Waaler. Optique 1.0: Semantic access to big data: The case of norwegian petroleum directorate's factpages. In E. Blomqvist and T. Groza, editors, *Proceedings of the ISWC 2013 Posters & Demonstrations Track, Sydney, Australia, October 23, 2013*, pages 65–68. CEUR-WS.org, 2013.

[KJZ+13] E. Kharlamov, E. Jiménez-Ruiz, D. Zheleznyakov, D. Bilidas, M. Giese, P. Haase, I. Horrocks, H. Kllapi, M. Koubarakis,

Ö. L. Özçep, M. Rodriguez-Muro, R. Rosati, M. Schmidt, R. Schlatte, A. Soylu, and A. Waaler. Optique: Towards OBDA systems for industry. In P. Cimiano, M. Fernández, V. Lopez, S. Schlobach, and J. Völker, editors, *The Semantic Web: ESWC 2013 Satellite Events - ESWC 2013 Satellite Events, Montpellier, France, May 26-30, 2013, Revised Selected Papers*, pages 125–140. Springer, 2013.

[KKPZ13] S. Kikot, R. Kontchakov, V. V. Podolskii, and M. Zakharyaschev. Query rewriting over shallow ontologies. In T. Eiter, B. Glimm, Y. Kazakov, and M. Krötzsch, editors, *Informal Proceedings of the 26th International Workshop on Description Logics, Ulm, Germany, July 23 - 26, 2013*, pages 316–327. CEUR-WS.org, 2013.

[KKZ11] S. Kikot, R. Kontchakov, and M. Zakharyaschev. On (in)tractability of OBDA with OWL 2 QL. In R. Rosati, S. Rudolph, and M. Zakharyaschev, editors, *Proceedings of the 24th International Workshop on Description Logics (DL 2011), Barcelona, Spain, July 13-16, 2011*. CEUR-WS.org, 2011.

[KKZ12] S. Kikot, R. Kontchakov, and M. Zakharyaschev. Conjunctive query answering with OWL 2 QL. In G. Brewka, T. Eiter, and S. A. McIlraith, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012*. AAAI Press, 2012.

[KLT+10] R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and M. Zakharyaschev. The combined approach to query answering in dl-lite. In F. Lin, U. Sattler, and M. Truszczynski, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference, KR 2010, Toronto, Ontario, Canada, May 9-13, 2010*. AAAI Press, 2010.

[Len11] M. Lenzerini. Ontology-based data management. In C. Macdonald, I. Ounis, and I. Ruthven, editors, *Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011, Glasgow, United Kingdom, October 24-28, 2011*, pages 5–6. ACM, 2011.

[Llo84] J. W. Lloyd. *Foundations of Logic Programming, 1st Edition*. Springer, 1984.

[LMR+14] D. Lembo, J. Mora, R. Rosati, D. F. Savo, and E. Thorstensen. Towards mapping analysis in ontology-based data access. In R. Kontchakov and M. Mugnier, editors, *Web Reasoning and Rule Systems - 8th International Conference, RR 2014, Athens, Greece, September 15-17, 2014. Proceedings*, pages 108–123. Springer, 2014.

[MM82] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Trans. Program. Lang. Syst.*, 4(2):258–282, 1982.

[PLC+08] A. Poggi, D. Lembo, D. Calvanese, G. D. Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. Data Semantics*, 10:133–173, 2008.

[PLL+13] F. D. Pinto, D. Lembo, M. Lenzerini, R. Mancini, A. Poggi, R. Rosati, M. Ruzzi, and D. F. Savo. Optimizing query rewriting in ontology-based data access. In G. Guerrini and N. W. Paton, editors, *Joint 2013 EDBT/ICDT Conferences, EDBT '13 Proceedings, Genoa, Italy, March 18-22, 2013*, pages 561–572. ACM, 2013.

[RKZ13] M. Rodriguez-Muro, R. Kontchakov, and M. Zakharyaschev. Ontology-based data access: Ontop of databases. In H. Alani, L. Kagal, A. Fokoue, P. T. Groth, C. Biemann, J. X. Parreira, L. Aroyo, N. F. Noy, C. Welty, and K. Janowicz, editors, *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*, pages 558–573. Springer, 2013.

[Rud11] S. Rudolph. Foundations of description logics. In A. Polleres, C. d'Amato, M. Arenas, S. Handschuh, P. Kroner, S. Ossowski, and P. F. Patel-Schneider, editors, *Reasoning Web. Semantic Technologies for the Web of Data - 7th International Summer School 2011, Galway, Ireland, August 23-27, 2011, Tutorial Lectures*, pages 76–136. Springer, 2011.

[W3C14] W3C. Owl 2 web ontology language profiles, October 2014. http://www.w3.org/TR/owl2-overview/.